Future Generation Computer Systems 40 (2014) 30–43

Contents lists available at ScienceDirect

Future Generation Computer Systems

journal homepage: www.elsevier.com/locate/fgcs

Towards high-available and energy-efficient virtual computing environments in the cloud

Altino M. Sampaio [a], Jorge G. Barbosa [b,*]

[a] Instituto Politécnico do Porto, Escola Superior de Tecnologia e Gestão de Felgueiras, CIICESI, Felgueiras, Portugal

[b] Universidade do Porto, Faculdade de Engenharia, Departamento de Engenharia Informática, LIACC, Porto, Portugal

HIGHLIGHTS

- A review of failure-aware algorithms for resource provisioning on clouds.
- The development of power- and failure-aware cloud scheduling algorithms that implement vertical and horizontal platform elasticity.
- Autonomous readjustment of virtual-to-physical mappings on-the-fly.
- Extensive evaluation of the dynamic strategy and of the scheduling algorithms.
- Random workloads and workloads that follow the Google cloud tracelogs.

ARTICLE INFO

Article history:
Received 1 August 2013
Received in revised form
25 June 2014
Accepted 27 June 2014
Available online 7 July 2014

Keywords:
Scheduling
Energy-efficiency
Consolidation
Proactive fault-tolerance
Platform elasticity

ABSTRACT

Empowered by virtualisation technology, cloud infrastructures enable the construction of flexible and elastic computing environments, providing an opportunity for energy and resource cost optimisation while enhancing system availability and achieving high performance. A crucial requirement for effective consolidation is the ability to efficiently utilise system resources for high-availability computing and energy-efficiency optimisation to reduce operational costs and carbon footprints in the environment. Additionally, failures in highly networked computing systems can negatively impact system performance substantially, prohibiting the system from achieving its initial objectives. In this paper, we propose algorithms to dynamically construct and readjust virtual clusters to enable the execution of users' jobs. Allied with an energy optimising mechanism to detect and mitigate energy inefficiencies, our decision-making algorithms leverage virtualisation tools to provide proactive fault-tolerance and energy-efficiency to virtual clusters. We conducted simulations by injecting random synthetic jobs and jobs using the latest version of the Google cloud tracelogs. The results indicate that our strategy improves the work per Joule ratio by approximately 12.9% and the working efficiency by almost 15.9% compared with other state-of-the-art algorithms.

1. Introduction

Cloud computing is a specialised distributed-computing paradigm that has recently gained popularity as a resource platform for on-demand, high-availability, and high-scalability access to resources. Cloud computing represents a new type of computational model, providing better use of distributed resources, while offering dynamic, flexible infrastructures and quality of service (QoS). From a hardware point of view, users have the illusion of infinite computing resources that are available on demand [1,2].

A cloud computing model exploits virtualisation [3] to render vertical and horizontal scalability, interoperability, failover, load balancing, improved system manageability, and reduction of infrastructure operational costs through resource consolidation. Virtual machine (VM) technologies provide flexible and scalable system services to cloud computing systems, creating a powerful computing environment where virtualised resources can be dynamically allocated, expanded, reduced, or moved as demand varies. With the development of virtualisation tools such as KVM, Xen, and VMware, new capabilities are introduced, such as VM live migration, VM checkpoint/restart, and VM pause/unpause, while the performance overhead for applications is maintained within an acceptable range [3,4].

As cloud computing follows a utility model of consumption, users contract computing power based on their expected needs.

* Corresponding author. Tel.: +351 220 413 279.
E-mail addresses: ams@estgf.ipp.pt (A.M. Sampaio), jbarbosa@fe.up.pt (J.G. Barbosa).

http://dx.doi.org/10.1016/j.future.2014.06.008
0167-739X/

However, not all users request the contracted computing power for the entire period of the contract. Meisner et al. [5] showed that much of the energy used in a data centre is wasted on idle systems, and, in typical deployments, server utilisation is below 30%. As energy costs are becoming a more significant factor in the operational costs of modern businesses, it is a natural consequence that energy consumption enters into the service cost equation to provide performance at reasonable prices. Rather than ensuring absolute performance at any cost, the focus is to maintain high service-level performance [6]. In this study, we measured the service performance by the success rate of job completion by the deadline. The service-level agreement (SLA), or contract, is defined for each job by specifying a deadline guaranteed by the system.

Node failures are a characteristic of a distributed system that can have thousands of nodes running a variety of different jobs. Fu [7] reported that a node's mean time between failures (MTBF) is 1.25 h in a petaflop system. In this paper, we propose a two-step strategy, composed of a cloud manager and a cloud scheduler, to construct and dynamically manage energy-efficient virtual clusters to execute sets of independent tasks in which computing resources are affected by failures. To improve the availability of virtual clusters and maximise the rate of completed jobs, we apply a proactive failure tolerance technique. We consider the problem of dynamically mapping tasks, running on virtual machines, to physical machines (PMs) in a power- and failure-aware manner. These virtual-to-physical resource-mapping decisions must consider the performance status, power efficiency, and reliability levels of the computing nodes. The optimisation decision includes selecting the CPU capacity necessary to accomplish the tasks within their respective deadlines, considering the predicted MTBF of each node and the migration overhead of the VMs.

By ensuring that tasks are completed by their deadlines, we are satisfying the SLA imposed on each job. However, we also consider that an SLA violation may occur, and, therefore, the SLA should include a penalty to the service provider for those occurrences, as proposed in [6].

Our optimisation strategy also enables the infrastructure to react to energy inefficiencies by continuously monitoring power consumption. Our approach is adequate to implement scalable and elastic service platforms on demand because of its dynamic characteristic. Additionally, it adapts to and reduces stop times in maintenance operations. In such situations, the administrators should only specify the PMs and their maintenance operation times. Our proposed strategies enable the autonomous rescheduling of the tasks currently running to complete the submitted users' jobs in an energy-efficient manner.

The main contributions of this paper are:

1. a review of failure-aware algorithms for resource provisioning of cloud infrastructures;
2. the development of power- and failure-aware cloud scheduling algorithms that implement vertical and horizontal platform elasticity;
3. the development of a dynamic scheduling strategy to provide power- and failure-aware virtual clusters that reactively detect energy optimising opportunities and perform consolidation to reduce energy consumption while maintaining service-level performance;
4. an extensive evaluation of the dynamic strategy and the scheduling algorithms with randomly generated workloads and with workloads that follow the latest version of the Google cloud tracelogs; and
5. a dynamic configuration of the CPU portion assigned to each VM that reduces the consumed energy and maintains the service-level performance.

The scheduling algorithms proposed in this study are improved versions of our previous work [8]. Additionally, we introduce an extended evaluation using randomly generated workloads and realistic workloads based on the recent Google cloud tracelogs [9].

The remainder of this paper is organised as follows. Section 2 introduces and discusses the related work based on dynamic placement and provisioning of VMs. Section 3 presents the architecture of our controller to enable a power- and failure-aware dynamic mapping of VMs to PMs, formulates the VM placement and provisioning problems, and introduces the proposed algorithms. Section 4 introduces the metrics used to evaluate the performance of the algorithms and describes the workloads and failure characteristics used in the simulations. Section 5 presents the results and discusses the performance of the proposed algorithms. Section 6 presents the conclusions of our current research and introduces future research directions.

2. Related work

This paper presents an approach for the dynamic scheduling of virtual machines in clusters, considering both optimisation of energy efficiency and physical-node reliability to provide energy-efficient and highly available computing environments. Below, we discuss relevant work in the literature related to similar issues.

The work by Xu and Fortes focused on initial [10] and dynamic [11] mapping of VMs to physical resources. The objectives in both studies were to simultaneously minimise costs due to total resource wastage, power consumption, and thermal dissipation. The first publication, on initial mapping, proposed an improved genetic algorithm with fuzzy multi-objective evaluation to efficiently search the large solution space and conveniently combine the conflicting objectives. The second publication concentrated on the dynamic placement of the VMs, where the authors aimed to optimise a multi-objective utility function that aggregates the three objectives in a weighted sum. The authors also defined threshold values and observation window sizes for condition detection and the stabilisation of the computing infrastructure. The PADD scheme [12] minimised energy consumption while satisfying service-level agreement (SLA) requirements. The authors' scheme dynamically migrated the VMs onto fewer PMs during periods of low utilisation and then expanded them onto more nodes if the need arose. The proposed algorithm aimed to minimise the total energy while meeting a given performance requirement and is based on the free CPU capacity thresholds of the nodes. It also uses a buffering technique to implement safety margins, thus reserving capacity to address workload demand fluctuations. The SLA specifies how much of a demand must be processed within a given time limit. SLA violations can occur because the PADD scheme is best-effort-based. Lee and Zomaya [13] proposed two consolidation task heuristics that maximise resource utilisation and explicitly take into account both active and idle energy consumption. The heuristics assign each task to the resource for which the energy consumption for executing the task is minimised without any performance degradation. Beloglazov et al. [6] proposed a modified best-fit algorithm for energy-aware resource provisioning in data centres while continuing to deliver the negotiated SLA. The performance is measured by the percentage of SLA violations that occurred. Unlike our work, none of the cited studies addressed the problem of scheduling with respect to energy efficiency and failure occurrence. However, the last study described above is used as a comparison baseline, where no failure-aware mechanism is implemented and the VM re-initiates when there is a failure. This is referred to as the common best-fit (CBFIT) algorithm in our results section and is further described in Section 3.6.

Fault-tolerant scheduling has been extensively studied in the literature regarding real-time systems, such as in Al-Omari et al. [14] and Zhu et al. [15], where fault tolerance is achieved using a primary backup technique that consists of scheduling the same task in two different processors; therefore, a failure can be tolerated, and the task is still able to execute before its deadline. This

technique privileges processing time without considering the energy consumed, and therefore, it is not considered here in more detail.

Various other studies have addressed the issue of computing environment availability and reliability, targeting SLA fulfilment. For example, Feller et al. built Snooze [16], a scalable, fault-tolerant, and distributed consolidation manager for heterogeneous clusters. The goal of Snooze is to dynamically consolidate the workload of the underlying heterogeneous software and hardware cluster arrangement, which is composed of a combination of both virtual and non-virtual machines. The approach divides the problem into two parts, namely, consolidation and idle-time management, and formulates the mapping of the workload to the PMs as an instance of a one-dimensional bin-packing problem (CPU) in which the PMs represent the bins and the workload represents the items to be packed. To solve this problem, the authors used heuristic algorithms with relaxed constraints that take into account the migration of the VMs. Additionally, they used the replication technique to achieve fault-tolerance, which increases the consumed energy to complete the same set of jobs. Loveland et al. [17] combined different virtualisation technologies to provide high-availability (HA) configurations based on redundancy (such as active/active and active/passive) while minimising costs. Nagarajan et al. [18] performed the first comprehensive study of proactive fault tolerance (FT) using VM migration mechanisms. The authors applied Xen to migrate an MPI task from a health-deteriorating node to a healthy node. Their solution integrates an intelligent performance monitoring interface (IPMI) for health enquiries (migrations are threshold-activated) with Ganglia, which determines node targets for migration based on load averages. Each node in the cluster runs a daemon that monitors resource usage and then multicasts that information to all the other nodes. Thus, all the nodes have an approximate view of usage throughout the entire cluster. The proactive FT daemon selects the target node for migration as the node that does not yet host a guest virtual machine and has the lowest CPU usage. The FT daemon provides three features, i.e., (i) health monitoring, (ii) decision making, and (iii) load balancing. In the case of a node failure without prior health deterioration symptoms, the system automatically reverts to the reactive fault tolerance approach, i.e., by restarting from the last checkpoint. The combination of proactive and reactive mechanisms decreases the cost of reactive fault tolerance by lowering the checkpoint frequency. The VgrADS project [19] provides a virtual grid execution system that provides uniform qualitative resource abstraction of aggregate resources from disparate sources under different policies, such as grids and clouds. The authors applied virtual grid execution for scheduling sets of deadline-sensitive weather forecasting workflows, balancing performance, reliability, and cost. Fault tolerance is achieved through the replication of task execution. The system works by first applying a rank value to each workflow job and then scheduling the jobs based on their rank priorities. Fu [7] investigated and proposed failure-aware node selection strategies for the construction and reconfiguration of virtual clusters to enhance system availability, achieving high performance. His approach leverages proactive failure management techniques, based on VM migrations, and considers both the performance and reliability status of computing nodes when making selection decisions. He proposed the optimistic best-fit (OBFIT) and pessimistic best-fit (PBFIT) algorithms to determine the best qualified nodes to which to allocate the VMs to run user jobs. Experiments showed that a higher rate of successfully completed jobs was achieved by using OBFIT and PBFIT strategies. However, these algorithms do not perform well with bad prediction accuracy (below 40%). The results showed that the approach enabled a 17.6% increase in job completion rate compared with that achieved with the current LANL HPC cluster. The algorithms OBFIT and PBFIT were proposed for the same scenario considered in this study, and they were then compared with our proposed power- and failure-aware scheduling algorithms.

3. Energy- and failure-aware virtual clusters

This section provides the formal description of an energy-efficient and failure-aware cloud architecture, which dynamically maps VMs to PMs, to improve the completion rate of users' jobs while decreasing energy consumption.

3.1. System overview and problem formulation

We consider a private cloud computing environment consisting of a cloud provider and multiple cloud users. The cloud architecture, information flow, and relative control blocks are illustrated in Fig. 1. The cloud computing infrastructure is composed of h physical machines, where M is the vector representing the PMs, $M = \{m_1, \ldots, m_h\}$. Physical hosts are homogeneous, i.e., they have the same CPU capacity C, memory capacity R, network bandwidth N, access to a shared storage space S for storing the disk images of the VMs, and predicted time in the future for the occurrence of failures F_i, which can vary among PMs such that $m_i = \{C, M, N, S, F_i\}$. In our work, we define a failure as any anomaly caused by hardware or software fault, an unstable environment, or intentional or mistaken actions by the infrastructure administrator that stops the computing infrastructure components from working correctly.

In a typical usage scenario, cloud users submit their jobs, and the cloud manager then reserves the necessary resources from the cloud infrastructure to run their jobs. Each job $j = (T_j, d_j)$ is composed of a set of n independent CPU-intensive tasks, $t_q \in T_j$, $q \in \{1, \ldots, n\}$, and a deadline, d_j. Once tasks are independent, the job deadline becomes the deadline of the longest task. The task workloads are expressed in Mflops. The cloud manager runs the resource allocation algorithm that creates mappings of tasks to machines. Then, it creates and manages VMs to execute the tasks, where each VM will run on top of one PM at a time. The set of VMs constitutes the user's virtual cluster execution environment. A job is scheduled only if a VM can be assigned to each task of the job. A VM encapsulates the task execution environment and is the unit of migration in the system. In our model, multiple distinct VMs can be mapped to a single PM. Job deadlines become activated as soon as cloud users submit jobs.

Although machines are composed of several components, such as memory, network interface cards, and storage disks, the power consumed by the active physical nodes is primarily dictated by the CPU resource [20,6]. Thus, we consider only the CPU power consumption in our energy model. Based on [11,6], the power consumption, P, of a PM, i, can be estimated based on the linear power model, $P_i = p1 + p2 \times CPU\%$, where $CPU\%$ is the percentage of CPU utilisation for a given time interval, measured for a PM i at runtime. The $p1$ and $p2$ factors are the power consumption when the PM is in idle mode and the additional power consumption from CPU utilisation, respectively. The factor $p2$ is typically proportional to the overall system load. Eq. (1) is the power efficiency of a PM i at a specific sample time and reflects how much useful work is produced for a given power consumption as follows:

$$E_{P_i} = \frac{CPU_i\%}{p1 + p2 \times CPU_i\%} \times (p1 + p2) \tag{1}$$

where the workload is represented by the percentage of CPU utilisation and the factor $p1 + p2$ is used to normalise the efficiency. The power efficiency increases monotonically with the workload, reaching 1 at 100% CPU usage [10,11]. By efficiently consolidating the VMs, idle PMs can go into sleep mode, which improves power consumption because an idle PM consumes from 60% to 70% of its total power [20,6].

The optimisation problem consists of scheduling all the tasks of the submitted jobs so that the energy consumed is minimised, with the realisation that the PMs are the cause of failures. Every

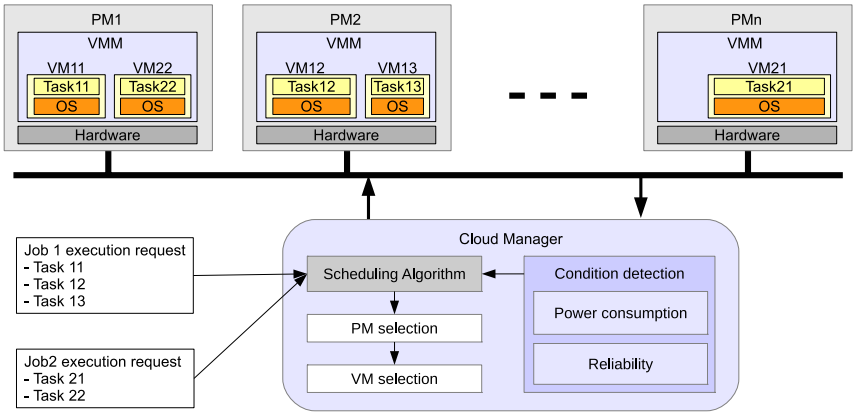


Fig. 1. Private cloud management architecture.

task t for all submitted jobs (J) may run on any machine $m \in M$. A schedule is defined by a function $sched_J : J \rightarrow M$ that assigns to each task $t \in J$ a set of machines $M_t \subset M$. Because of node failures, each task may be executed on a single machine or may migrate between machines more than once. The failure prediction mechanism specifies the off times for each machine, and a schedule for a task may specify a set of machines on which the tasks are separately executed.

The objective function is the maximisation of the average power efficiency of the execution of all jobs, defined by Eq. (2), for all active physical nodes u, at all sample times f, as follows:

$$\overline{E_P} = \frac{\sum_{s=1}^{f} \frac{\sum_{i=1}^{u} E_{P_i}}{u}}{f}, \quad \forall u \leq h \tag{2}$$

subjected to the deadline constraint given by Eq. (3) for each task t as follows:

$$FT(t) \leq d_t \tag{3}$$

where FT is the completion time of the task and d_t is the deadline of t.

The deadline constraint may lead to an empty schedule depending on the resources available, their reliability, and the difference between each task execution time and its deadline. The formulation presented above considers a set of jobs available at the scheduling time. However, the cloud is dynamic with respect to resource availability and number of job submissions by users. Thus, a new schedule would have to be produced when any of those conditions change. The optimisation problem is NP-complete and cannot be readily solved in a dynamic environment. Therefore, it is important to develop heuristic algorithms, such as those proposed in this paper, which can adapt the schedules on-the-fly.

Resource sharing is a common practice in virtualised environments. However, despite hypervisor slicing and resource sharing among co-located VMs, the execution of one VM can still affect the performance of the others. This phenomenon is known as performance interference and is caused by the competition for resources by running tasks. Performance interference in virtual environments induces slowdown of running applications, thus imposing a deviation between expected and delivered QoS. Several authors have investigated this problem and have proposed techniques to predict the slowdown [21], where performance degradation, from the last level of cache space and memory bandwidth sharing, can be estimated with an error less than 4%. In this study, the SLA was based on the specific deadline for each job; therefore, we can accommodate the estimated performance degradation when selecting the PMs and assigning a CPU percentage to each task. In our model, without loss of generality, we assume that there is no degradation when resources are shared to evaluate the proposed algorithms with respect to energy management and job completion rates during node failure without introducing additional variables.

3.2. Cloud manager

The cloud manager continually obtains the virtual and physical machine statuses, such as the nodes' reliability, power consumption, and execution progress of the tasks. Based on the information collected, the cloud manager makes decisions concerning opportunities to improve energy efficiency, e.g., VM consolidation or the need to migrate VMs to facilitate PM failure toleration. The consolidation mechanism transfers VMs from lower, loaded PMs to other PMs, thereby improving the energy efficiency by putting the first PMs in sleep mode and increasing the load rate of the active PMs.

Forecasting the time at which the next failure is going to occur, in a certain PM, can be determined using the tool described in [7,22]. The authors argued that the predictor tool can accurately predict failure occurrences with an average accuracy of 76.5%. If a PM fails unpredictably or fails before its running VMs conclude the migration process, those VMs will be re-initiated in spare PMs.

3.2.1. PM states

Because a machine in an idle state can consume from 60% to 70% of the power consumed when it is running at full CPU speed [20,6], unused systems can enter a sleep state [23] to reduce power consumption. There have been some notable efforts to reduce the overhead caused by entering and waking from a sleep state. For example, it has been shown [5] that a typical blade server consuming 450 W at peak can transit rapidly in a fraction of a millisecond to a near-zero-power idle state, consuming approximately 10.4 W. Thus, a reduction of power consumption can be effectively achieved in modern real-world systems by switching idle PMs to sleep mode in response to instantaneous loads, incurring near-zero overhead. Another feasible technique to reduce the power consumption of idle, or low-loaded, PMs is to use Dynamic Voltage Frequency Scaling (DVFS) [24]. However, as is noted in [6], DVFS is only applied to the CPU and not to other system components, resulting in lower gains of energy compared with the technique of switching idle PMs to the sleep state.

3.2.2. VMs migration

Migrations are required either for energy optimisation or when a failure is predicted to occur. We opted for stop and copy migrations that accomplish the migration process in a shorter interval of time than live migrations [25]. In a live migration, memory pages are iteratively copied from source to destination node, without stopping the execution of the migrating VM [26]. The overhead introduced by this approach is a function of the frequency

Algorithm 1 Cloud manager algorithm

```
1: function cloudManager(pmList, cloudUsers)
2:     Event e ← NULL
3:     while true do
4:         jobList ← NULL
5:         if e == ConsolidationEvent then
6:             jobList.add(pmList.getTasksFromPMsPowerInefficient(γ, τ))
7:         else
8:             jobList.add(pmList.getTasksFromPMsAboutToFail(ζ))
9:             jobList.add(pmList.getTasksFromPMsFailed())
10:            jobList.add(cloudUsers.getUnscheduledJobs())
11:        end if
12:        map ← schedAlg(pmList, jobList.getTaskList())
13:        if map ≠ NULL then
14:            executeMapping(map)                    ▷ Executes the VMs to PMs mapping
15:            pmIdleList ← NULL, map ← NULL
16:            pmIdleList.add(pmList.getIdlePMs())
17:            pmSetSleepMode(pmIdleList)
18:        end if
19:        e ← WaitForEvent()
20:    end while
21: end function
```

of the writing to the memory pages because dirty memory pages must be re-copied. In contrast, stop and copy migration requires the VM to be stopped and its image to be transferred before execution restarts at the new location. In this case, migration time depends on the amount of memory allocated to the VM, which is completely transferred; thus, the cost is mostly constrained by the network bandwidth [6,27].

Several strategies for VM migration have previously been evaluated [18,25] with respect to working set size, and the results have shown that live migration of VMs can almost double the migration time compared with the time associated with the migration of VMs using the stop and copy method. The migration of VMs with the stop and copy method requires 12–14 s, in contrast to the 14–25 s required for a live migration, on average.

3.2.3. Cloud manager algorithm

Algorithm 1 describes the cloud manager algorithm. In line 8, the PM failure prediction mechanism identifies the PMs that are about to fail and forces all VMs running on those machines to migrate before a failure occurs in the preservation of the work already completed. In line 9, the VMs that were running on PMs that failed, before migrating their VMs, are added to the job list to be initiated again. In line 10, all tasks that have not yet been scheduled in the last scheduling instant and tasks from the new jobs that, meanwhile, have arrived are added to the job list. In line 6, if there are no other tasks to schedule, we use the consolidation mechanism for power efficiency improvement [28] that uses a single threshold to identify under-utilised PMs. This mechanism identifies the tasks that, if migrated, would increase the system power efficiency and adds those tasks to the list. The destination PM is decided by the scheduling algorithm in line 12. If there are other tasks to schedule, the consolidation is postponed because those tasks will change the load on the PMs.

In line 12, a scheduler algorithm is called to map the set of available tasks to the PMs, regardless of whether they are new tasks or tasks to be migrated. Only tasks that can start immediately are mapped. The remaining tasks stay on the unscheduled list, i.e., the result produced from the scheduling algorithm is not a plan for mapping all available tasks. Such an approach enables the scheduling algorithm to execute faster and is more adapted to the dynamic behaviour of a cloud.

If a valid map results from the scheduling algorithm, it is applied in line 14. The map includes the initiating of new tasks, as well as the migration of running tasks. If more than one migration occurs from a source PM, they occur sequentially in time. After the map is applied, the idle machines are collected in line 16 and are set to sleep mode in line 17. Then, the algorithm waits for an event in line 19. The events that may occur are changes to the environment, namely, a PM failure, a PM failure estimate, a VM consolidation event, completion of a task, or the arrival of a new job. When any of these events occurs, the cloud manager starts a new scheduling iteration. The assignment of the CPU portion assigned to a VM may be changed only if it is rescheduled from a migration operation or a task re-initiation.

3.3. Power- and failure-aware scheduling algorithms

In this subsection, we introduce our proposed scheduling algorithms to allocate VMs in a power- and failure-aware manner. The cloud manager invokes these algorithms to carry out task (re)scheduling. Because the problem of mapping the VMs to the PMs is NP-complete, the proposed algorithms are heuristic.

The submitted jobs and their tasks have heterogeneous resource requirements and varying deadlines. First, we define each task's minimum and maximum execution speeds and slack time, which are used by the algorithms to prioritise tasks.

Eq. (4) defines the minimum resources $r(t)$, in Mflops/s, assigned to a task t that are necessary to complete its remaining workload $W(t, \mu)$ in the time from time $\mu + m_{ei}$ to the task deadline d_t. For example, consider the matrix multiplication algorithm that requires $2n^3$ flops for matrices of size (n, n) to compute. If the amount of time available to execute the task is 120 s, the required resource is $2n^3/120$ flops/s. Likewise Stillwell et al. [29], we assume that a task t only requires the maximum amount of resources $\max r(t)$ necessary to execute at the maximum speed, a value defined by the user.

$$\min r(t) \geq \frac{W(t, \mu)}{d_t - \mu - m_{ei}} \tag{4}$$

where μ is the current time and the parameter m_{ei} represents the overhead required for a VM to migrate from node e to node i. The parameter m_{ei} is 0 if we are scheduling the task for the first time or re-initiating it.

The slack time of task t, as shown in Eq. (5), expresses the difference between the task deadline d_t and the minimum time

Table 1
List of symbols.

Symbol	Description
h	Number of physical servers/nodes
C_i	Capacity of physical node i
J_S, J_C	Numbers of jobs submitted and jobs completed, respectively
f	Number of time units for the simulation period
u	Number of active PMs, at a certain time unit s
$r_i(t)$	Task t capacity request over PM i (Mflops/s)
δ_i	Predicted instant of failure for node i
d_t	Completion deadline for task t
μ	The current time
$W(t, \mu)$	Remaining workload for task t, at instant μ (Mflops)
m_{ei}	Overhead migration of a VM/task, from node e to node i
R_i	Reliability weight of node i

necessary for task completion as follows:

$$slack\_time = (d_t - \mu) - \frac{W(t, \mu)}{\max r(t)}. \tag{5}$$

The tasks for which Eq. (5) returns a negative value are cancelled, and the corresponding jobs are considered incomplete, incurring an SLA violation. Assigning each task a CPU power between $\min r(t)$ and $\max r(t)$, we are satisfying the strict SLAs imposed on each job.

We configure the cap parameter [41] from the Xen credit scheduler for fine-grained CPU assignment, allocating the strictly necessary amount of CPU power to execute a task before its deadline. The Xen credit scheduler is the default CPU scheduler used by the Xen hypervisor [30]. It assigns a cap and weights to each VM instance to control the sharing of computing resources among several VMs. The weights determine the proportion of CPU time allocation to each VM when several VMs compete for the CPU, meaning that VMs with higher weights will have a higher execution priority. The cap specifies the maximum percentage of CPU resources that a VM can obtain in a non-work-conserving mode (i.e., a VM instance cannot use more than its share of the CPU). By dynamically adjusting the CPU cap using the Xen credit scheduler, the fraction of CPU resources assigned to each VM can be updated to explore the available CPU fraction in the physical node until the sum of the caps of all co-located VM instances is 100% [31].

In both algorithms proposed in this study, when a VM migration occurs, the CPU resource assigned to the VM is re-evaluated and the destination PM and VM requirements are adjusted; thus, the CPU requirement of the source PM may be different from that of the destination PM. This dynamic CPU update only occurs when migrations are performed, allowing horizontal and vertical scaling of the VMs with respect to the CPU allocation. Table 1 lists the symbols used in the schedule algorithms described next.

3.4. POwer- and Failure-Aware Relaxed time Execution (POFARE)

For the incoming jobs $\{j_1, \ldots, j_k\}$, the cloud manager creates a set of virtual clusters, one for each user, to execute the tasks associated with the jobs. When invoked by the cloud manager, the algorithm selects, at each step, a VM to schedule and determines a destination PM.

3.4.1. VM selection

The algorithm creates a list of prioritised groups of tasks $\{t_1, \ldots, t_{n\times k}\}$. The priorities are, from the highest to the lowest priority (1) proactive failure tolerance; (2) re-initiating of failed tasks; and (3) scheduling of new tasks. Power-optimising scheduling is performed when there are no other types of tasks. Then, the tasks in each group are sorted in ascending order according to the *slack_time* of the tasks (Eq. (5)). These steps correspond to lines 3–5 of the algorithm POFARE that is described in Algorithm 2.

3.4.2. VM placement

After a VM is selected to be scheduled from the prioritised list, in line 6 of Algorithm 2, all PMs are evaluated with respect to task resource requirements, PM power efficiency, and reliability. In line 11, the algorithm checks that node i can supply the minimum resources required by task t, which is evaluated using Eq. (6) as follows:

$$r_i(t) + \sum_{k=1}^{n} r_i(t_{i_k}) \leq C_i \tag{6}$$

where $\{t_{i_1}, t_{i_2}, \ldots, t_{i_n}\}$ is the set of tasks running on node i and $r_i(t_{i_k})$ is the resource assigned to task t_{i_k}. The machine is considered a candidate if the minimum resource demand of t, plus the overall resources required by all tasks that run on physical node i, does not exceed the node capacity C_i. For the first evaluation, $r_i(t)$ is $\min r(t)$.

If the first condition is true in line 11, the second condition is assessed, i.e., the algorithm verifies that the node i can execute part of the task before a failure in i occurs, as evaluated by Eq. (7). This equation indicates that task t can be (re)scheduled or migrated from node e to a non-reliable physical node i at time $\mu + m_{ei}$, which is predicted to fail at instant δ_i, if (1) it provides the resources $r_i(t)$ required by task t during the $\delta_i - \mu$ time interval and (2) task t will have to migrate to another, currently unknown physical node where it cannot require more resources than those needed to enable execution at maximum speed, $\max r(t)$.

$$\begin{aligned} &r_i(t) \times (\delta_i - \mu - m_{ei}) + (\max r(t)) \times (d_t - \delta_i - m_{ei}) \\ &\quad \geq W(t, \mu), \\ &\text{if } \Bigg((\delta_i - \mu - m_{ei}) \\ &\quad \leq \frac{W(t, \mu)}{\max r(t)} \wedge r_i(t) \in [\min r(t), \max r(t)] \Bigg). \end{aligned} \tag{7}$$

If the machine is not selected by the above equation, Eq. (8) is evaluated. In other words, if task t starts at time $\mu + m_{ei}$ at node i, with a predicted instant of failure δ_i later than the task deadline d_t, the amount of resources necessary to complete the task within the deadline can be allocated without additional migrations. If node i does not satisfy Eq. (8), it is discarded to receive the current task.

$$\begin{aligned} &r_i(t) \times (d_t - \mu - m_{ei}) \geq W(t, \mu) \\ &\text{if } \Bigg((\delta_i - \mu - m_{ei}) \\ &\quad > \frac{W(t, \mu)}{\min r(t)} \wedge r_i(t) \in [\min r(t), \max r(t)] \Bigg). \end{aligned} \tag{8}$$

All machines are evaluated and, in line 21, a maximum of one machine is the *pmSelected*, i.e., the node that best improves the power efficiency of the machine (Eq. (1)) and that can provide the required resources. If there is more than one node providing the same power efficiency, E_P, the node with the highest reliability is selected in line 14. Eq. (9) represents the reliability weight for a given node i. The node with the lowest R_i is the most reliable.

$$R_i = \frac{1}{2^{\delta_i - d_t - m_{ei}}}. \tag{9}$$

The criterion for choosing the node that returns the lowest value of Eq. (9) allows tasks that have a low slack time to be assigned to run in nodes that have the necessary resources to avoid migrations. The algorithm sets the cap parameter in the Xen credit scheduler to reserve the resources necessary to complete the tasks within their deadlines. The algorithm supports a work-conserving mode during task execution if the CPU capacity of a node is not completely consumed. In such cases, the remaining free CPU capacity is distributed equally among all running VMs until the entire

Algorithm 2 POwer- and Failure-Aware Relaxed time Execution (POFARE)

```
1:  function POFARE(pmList, jobList)
2:      mappingList ← NULL
3:      jobList.taskList.removeTasksHavingNegativeSlackTime()
4:      jobList.taskList.groupTasksByReason()
5:      jobList.taskList.sortTasksInGroupsByIncreasingSlackTime()
6:      for all task ∈ jobList.taskList do
7:          lRweight ← NULL                                   ▷ lowest reliability weight
8:          hEpower ← NULL                                    ▷ highest power efficiency
9:          pmSelected ← NULL
10:         for all pm ∈ pmList do
11:             if pm.hasResources(task) and pm.hasReliability(task) then
12:                 rWt ← pm.getReliabilityWeight(task)
13:                 ePw ← pm.getPowerEfficiency(task)
14:                 if ePw > hEpower or (ePw = hEpower and rWt < lRweight) then
15:                     lRweight ← rWt
16:                     hEpower ← ePw
17:                     pmSelected ← pm
18:                 end if
19:             end if
20:         end for
21:         if pmSelected ≠ NULL then
22:             mappingList ← {task, pmSelected}
23:             pmSelected.updateResourcesAvailable(task.minResources)
24:         end if
25:     end for
26:     return mappingList                                    ▷ VMs (tasks) to PMs mapping
27: end function
```

CPU capacity of the node is consumed or the maximum capacity of all VMs is reached, thus enabling the task to run at the maximum possible speed. In cases where there is no physical node providing the minimum required resources $\min r(t)$ to complete a task t by its deadline d_t, that task is put on hold until the cloud computing environment releases the necessary resources.

3.5. POwer- and failure-aware minimum time execution (POFAME)

The POFAME algorithm differs from the POFARE algorithm in the amount of CPU resources that are assigned to a task. POFAME selects a PM that maximises the power efficiency, as POFARE does, but it reserves the maximum amount of resources needed to execute the task (in line 23 of Algorithm 2), limited by the available PM resources and the maximum amount of resources required by the task. POFARE reserves the minimum required resources to complete the task by its deadline, thereby increasing the task's completion time.

The complexity of POFARE and POFAME is given by the product of the number of tasks, n, to be scheduled at a given time and the total number of machines, m. For a single iteration, the complexity of both algorithms is $O(n \times m)$.

3.6. Comparison of scheduling algorithms

We implemented three other algorithms to compare their performance with that of our algorithm, namely, the optimistic best-fit (OBFIT) algorithm, the pessimistic best-fit (PBFIT) algorithm [7], and a best-fit type of algorithm that is based on the MBFD algorithm [6] and herein called the common best-fit (CBFIT) algorithm. Because MBFD does not consider node failures, CBFIT is a simplified version to show the impact of the pro-active fault-tolerant migrations. The CBFIT strategy selects, from all available PMs, the PM that has the minimum capacity necessary to run a task to optimise energy consumption. In turn, OBFIT collects a set of PMs that will not fail before a task's deadline. Then, from this set of PMs, it weighs and selects the PM that has both the minimum required capacity and minimum reliability to run a VM. The PBFIT strategy calculates the average available capacity level $C_{average}$ among the computing nodes that will not fail before the task deadline, and from the set of PMs that will fail before the task deadline, it selects the PM with capacity C_p such that $C_{average} + C_p$ results in the minimum required capacity to run the task.

4. Evaluation and simulation scenario

In this section, the metrics used to evaluate the algorithm performance are presented. The workloads, failure characteristics, and simulation scenario are also described.

4.1. Performance metrics

To evaluate the performance of the algorithms, we defined three metrics: (i) the completion rate of users' jobs; (ii) the ratio of useful Mflops processed to the energy consumed; and (iii) the working efficiency. The first metric, expressed as Eq. (10), measures the completion rate of users' jobs, E_J, which is calculated as the ratio of the number of jobs completed by their deadline, J_C, to the number of submitted jobs, J_S. Its value falls in the interval $[0, 1]$, and it is the SLA metric. The difference between E_J and 1, multiplied by 100, is the percentage of SLA violations.

$$E_J = \frac{J_C}{J_S}. \tag{10}$$

The energy-efficiency metric, E_M, shown in Eq. (11), calculates the amount of energy consumed, in Joules, to produce useful work. By useful work, we count the number of Mflops associated with successfully completed jobs only, J_C. E_M is calculated by dividing the sum of the workloads from all tasks of successfully completed jobs by the overall energy consumption. The energy is then calculated by multiplying the average power consumption of the computing infrastructure (i.e., for all active physical nodes u at

all sample times f) by the number of sample times f, multiplied by 60 (because the samples are obtained each minute). We henceforth represent this metric as Mflops/Joule.

$$E_M = \frac{\sum_j \left(\theta_j \times \sum_{t=1}^{n} W(t,0)\right)}{\sum_{s=1}^{f} \frac{\sum_{i=1}^{u} P_i}{u}} \times f \times 60, \tag{11}$$

$$\theta_j = \begin{cases} 1, & \text{if job } j \text{ completed} \\ 0, & \text{otherwise.} \end{cases}$$

Eq. (12) shows the calculation of the working efficiency, E_W, which is used as a metric with which to determine the quantity of useful work performed (i.e., the completion rate of users' jobs) by the consumed power. It is determined by multiplying E_J, the completion rate of jobs, by the average power efficiency based on Eq. (1) for all active physical nodes $i \in [1, u]$ at all sample times f.

$$E_W = \frac{\sum_{s=1}^{f} \frac{\sum_{i=1}^{u} E_{P_i}}{u}}{f} \times E_J, \quad \forall u \leq h. \tag{12}$$

Eqs. (11)–(12) express the amount of useful work performed from different perspectives. The first equation quantifies the number of useful Mflops by the consumed energy, while the second equation measures the quantity of useful work (i.e., completion rate of users' jobs) performed with the consumed power. The best algorithm should be the algorithm that maximises both, enabling the processing of more Mflops with a lower amount of energy and maximising the job completion rate while keeping high levels of power efficiency.

4.2. Workload and failure characteristics

In this section, we describe the characteristics of the workloads and failures injected in the simulator to evaluate the performance of the scheduling algorithms.

4.2.1. Random synthetic workloads

To create a set of synthetic workloads, we chose the Poisson distribution as the basis of the synthetic jobs [32]. The average job inter-arrival time was set to 10 min, and each job was composed of an average of 10 tasks. Additionally, the average task length to MTBF ratio varied on a logarithmic scale of {0.01, 0.1, 1, 10}. In other words, if we consider a ratio of 1, the average task length equals the MTBF. The average CPU utilisation per task was set to 20% of the node capacity. The injection of this type of workload into the simulator will enable an analysis of the impact of the average task length to the MTBF ratio on the performance of the scheduling algorithm. Considering these workload characteristics, we created 100 synthetic jobs. A job deadline equals the deadline of its longest task, and the task deadlines are rounded up to 10% more than their minimum necessary execution time.

4.2.2. Workloads based on Google cloud tracelogs

Recent studies analysing the latest version of the Google cloud tracelogs [9], spanning 29 days, yielded significant data on the characteristics of submitted workloads and the management of cluster machines. These studies enable further work on important issues, such as resource optimisation, energy-efficiency improvements, and failure correlation. Some authors [33,34] argue that approximately 75% of jobs only run one task and most of the jobs have less than 28 tasks that determine the overall system throughput. The medium length of a job is 3 min, and the majority of jobs run in less than 15 min, despite the fact that there are a small number of jobs that run longer than 300 min. Moreover, task length follows a lognormal distribution [35], with most of the tasks requiring a short amount of time [36]. This same distribution applies to CPU usage, which varies from near 0% to approximately 25%, indicating that a high proportion of tasks consume resources at lower rates. For the same reason, a lognormal distribution can be applied to describe the number of tasks per job. Depending on the cluster or day observed, job inter-arrival times follow distributions such as lognormal, gamma, Weibull, or even exponential [34–36], with a mean time of 4 s. Regarding RAM usage, most of the tasks use less than 2.5% of the node's RAM [34]. Based on these studies, we created 3614 synthetic jobs to simulate cloud users' jobs, requiring a RAM size of 256, 512, or 1024 MB, selected randomly. The total number of tasks is 10 357, and each task deadline is rounded up to 10% more than its minimum necessary execution time. The job deadline equals the deadline of its longest task.

4.2.3. Failures and unavailability properties

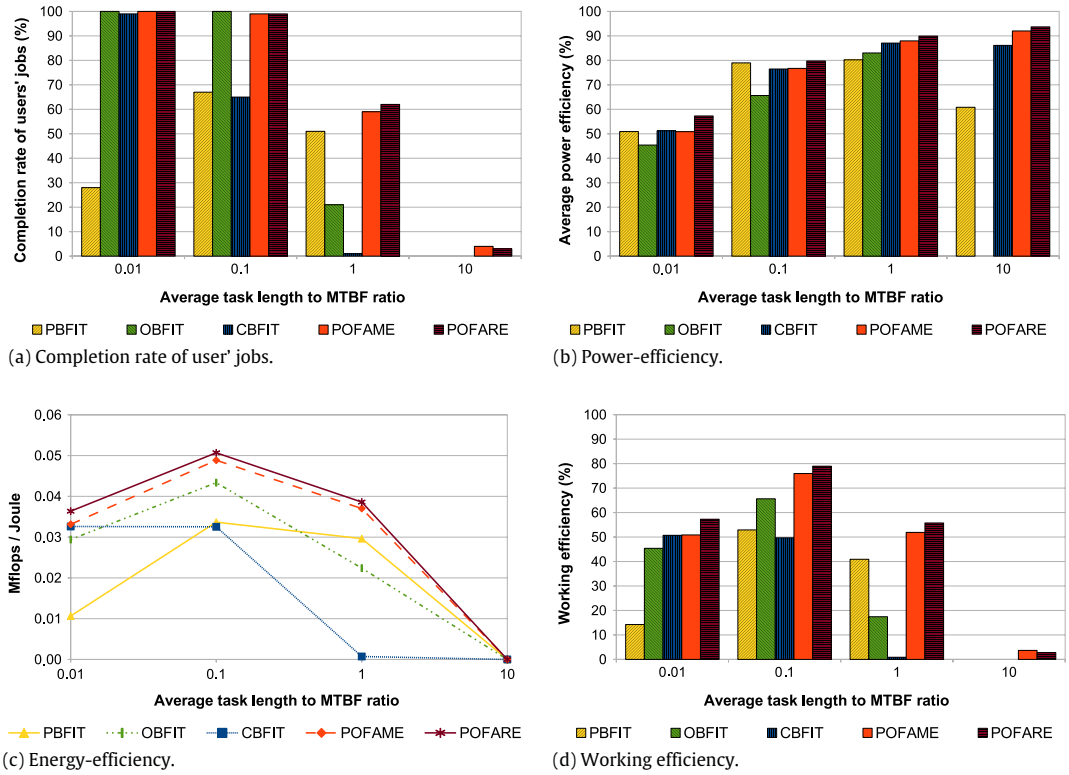
For each physical node, the MTBF is programmed according to a Weibull distribution, with a shape parameter of 0.8, which has been shown [37] to well approximate the time between failures for individual nodes, as well as for the entire system. Failed nodes stay unavailable (i.e., mean time to repair (MTTR)) during a period modelled by a lognormal distribution, with a mean time set to 20 min, varying up to 150 min. Failure tolerance is implemented through proactive VM stop and copy migration, rather than checkpointing. The predicted occurrence time of failure is earlier than the actual occurrence time. When a node fails, the tasks running on it are restarted in a different set of nodes, from scratch, if there is adequate time to execute those tasks before their deadlines. We have assumed that the initiating of VMs uses negligible overhead. Cavilla et al. [38] demonstrated that it is possible to initiate multiple VM instances in less than 1 s, assuming that the VM images and data are stored in a Network Attached Storage (NAS).

4.3. Simulation setup

We simulated the cloud computing infrastructure described in Fig. 1, which is composed of 50 homogeneous physical nodes. The CPU capacity of the physical nodes was assumed to be 800 Mflops/s, values that can be evaluated using Linpack [39]. The power consumed by the fully loaded physical nodes was 250 W. The parameters $p1$ and $p2$, for Eq. (1), were set to 70% and 30% of full power consumption, respectively. The scheduling algorithms have no knowledge of when jobs arrive. The average amount of CPU resources required by a VM was set to 160 Mflops/s (20% of a node's capacity) for the random workload, implying that, on average, the maximum number of VMs per node is 5. Each VM requires a RAM size of 256, 512, or 1024 MB, randomly selected. The migration overhead of the VM depends on the memory size and the network bandwidth, which, in this experiment, was set to 1 Gigabit/s [6,27]. Tasks with deadlines extending past the failure time of their node migrate $\zeta = 3$ min before the nodes predicted failure time.

When the consolidation mechanism is applied, the size of the sliding window with which to detect energy-optimising opportunities is 5 min, with a CPU usage threshold of $\tau = 55\%$ and number of occurrences within the window of $\gamma = 3$. We have chosen these values based on our previous study [28], which demonstrated that these values produce an optimal ratio of the amount of work performed to the consumed energy.

It has been shown elsewhere [7,22] that the failure predictor tool can predict failure occurrences with an average accuracy of 76.5%. In our simulations, we have measured the impact of failure prediction accuracy on the performance of the scheduling algorithms, as well as the average task length to MTBF ratio, when the prediction accuracy is 75%.



(a) Completion rate of user' jobs. (b) Power-efficiency.

(c) Energy-efficiency. (d) Working efficiency.

Fig. 2. Impact of the average task length to MTBF ratio in the performance of scheduling algorithms, without dynamic consolidation, for random workloads. The configuration is: (i) MTBF = 200 min; (ii) average failure prediction accuracy = 75%.

5. Results and analysis

This section presents the results associated with the algorithm performance. First, we inject a set of random workloads to investigate how well the algorithms perform for tasks with different average task length to MTBF ratios. Then, we use the Google-based workloads to assess the algorithm performance with and without the consolidation mechanism that dynamically re-adapts the schedule to improve power efficiency. The section ends with the reporting of preliminary results from the adoption of the proposed algorithms in a real platform.

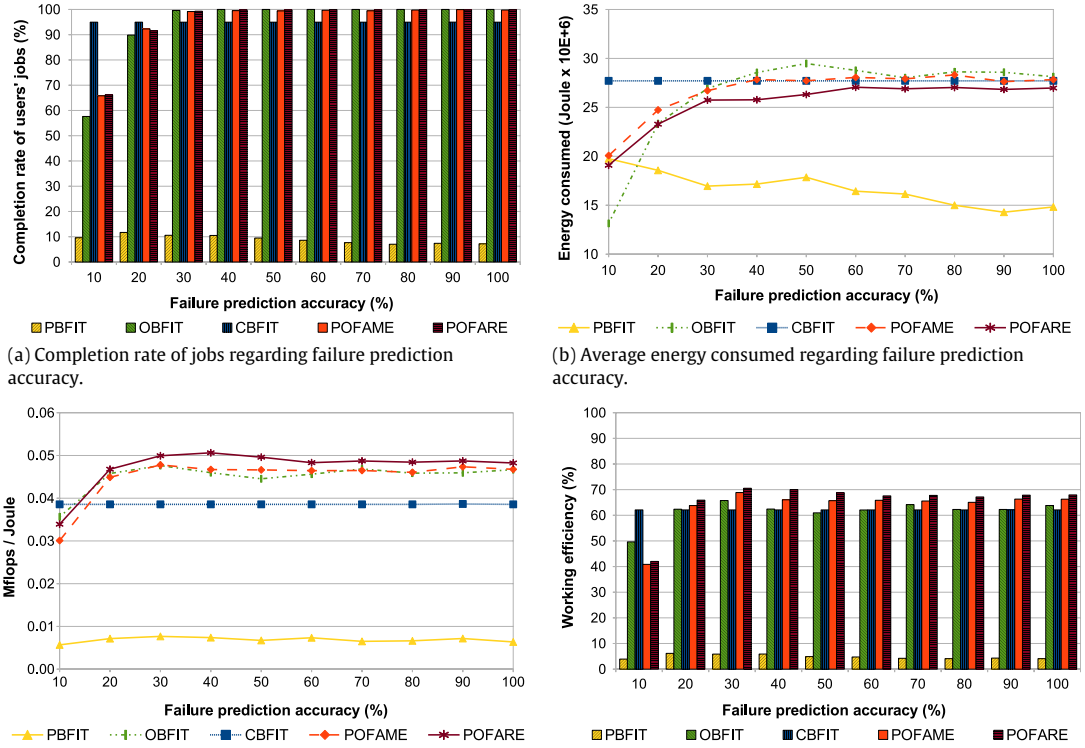
5.1. Random synthetic workloads

The results from the first set of simulations are shown in Fig. 2. These simulations used the synthetic workload characteristics introduced in 4.2.1. We fixed the MTBF at 200 min, and the average task length to MTBF ratio was varied within the set {0.01, 0.1, 1, 10}. Additionally, the average prediction accuracy for the failure predictor tool was set at 75%, and the results were obtained without dynamic consolidation. The CBFIT algorithm does not consider failures when scheduling and does not perform migrations when failures occur. It re-initiates the tasks of a node when it fails. CBFIT is a best-fit algorithm that is used in this study as a basis for performance comparison.

Fig. 2(a) shows the job completion rate as a function of the average task length to MTBF ratio. It shows that as the average task length increases compared with the MTBF, the completion rate of users' jobs decreases. For the values of 0.01 and 0.1, task lengths are small compared with the MTBF and OBFIT, POFAME and POFARE perform similarly, completing almost 100% of the jobs. CBFIT also performs well for 0.01 because the task lengths are small compared with the MTBF; thus, the failures do not affect its performance. However, for the 0.1 case, the job completion rate for CBFIT decreases significantly. For the values of 1 and 10, failures occur in almost all tasks and CBFIT completes only 1% of tasks for a ratio of 1 and 0% for a ratio of 10, with most of the time spent on re-initiating tasks. For a given task, OBFIT only selects nodes that do not fail until the task deadline, and, therefore, it does not perform migrations to prevent failures. Consequently, its performance also decreases significantly for the ratio of 1, with only 21% of the jobs completed. For the ratio of 0.1, POFARE and POFAME complete 99% of the jobs. Although it is a high rate of success, it is not 100% because of the additional overhead caused by the migration of the VMs to tolerate node failures. The migration technique shows significant improvements for the ratios of 1 and 10 compared with the other algorithms. In contrast, PBFIT tends to complete more jobs when dealing with unreliable PMs, as it only schedules tasks to unreliable PMs.

Fig. 2(b) shows the average power efficiency that allows the evaluation of the consolidation rate achieved by each algorithm. For the average task length to MTBF ratio of 0.01, low power efficiency is achieved, i.e., approximately 50%, meaning that the load of a node is far from 100%. As the task sizes increase, the nodes run at a higher percentage of CPU utilisation. POFARE is the best algorithm in all cases; the results are above 90% for the ratios 1 and 10. The power efficiency indicates the node loads during the experiments, including the workload of tasks that could not be completed by their deadlines. For a ratio of 10, OBFIT obtains a power efficiency of 0 because it does not schedule tasks on unreliable PMs, and, therefore, it does not launch any VMs. The other algorithms achieve high rates of power efficiency, but only POFAME and POFARE are able to complete jobs at rates of 4% and 3%, respectively.

Fig. 2(c) illustrates the energy efficiency (E_M), which is represented by the ratio of the useful work to the energy consumed (Mflops/Joule). For the average task length to MTBF ratio of 0.01, all algorithms achieve a Mflops/Joule ratio lower than that for the ratio of 0.1 because the same job completion rate is achieved in both cases (Fig. 2(a)), but the consolidation is lower for the ratio of 0.01 (Fig. 2(b)). The highest rate of Mflops/Joule is achieved for 0.1 and decreases for the ratios of 1 and 10 for all algorithms. The



(a) Completion rate of jobs regarding failure prediction accuracy.

(b) Average energy consumed regarding failure prediction accuracy.

(c) Energy-efficiency regarding failure prediction accuracy.

(d) Working efficiency regarding failure prediction accuracy.

Fig. 3. Energy- and failure-aware scheduling algorithms, without dynamic consolidation, for Google-based workloads. The configuration is: (i) average task length = 3 min; (ii) MTBF = 200 min.

CBFIT algorithm yields a Mflops/Joule value near zero for the ratio of 1 because the job completion rate is only 1%. POFARE shows an improvement over OBFIT by 23.6%, 16.9%, and 72.4% for the ratios of 0.01, 0.1, and 1, respectively.

The working efficiency metric is shown in Fig. 2(d). It measures the useful work based on the consumed power, similar to the energy efficiency, which considers energy; therefore, the same function shape is obtained as in Fig. 2(c). POFARE shows an improvement over OBFIT of 26.2%, 20.3%, and 219.7% for the ratios of 0.01, 0.1, and 1, respectively.

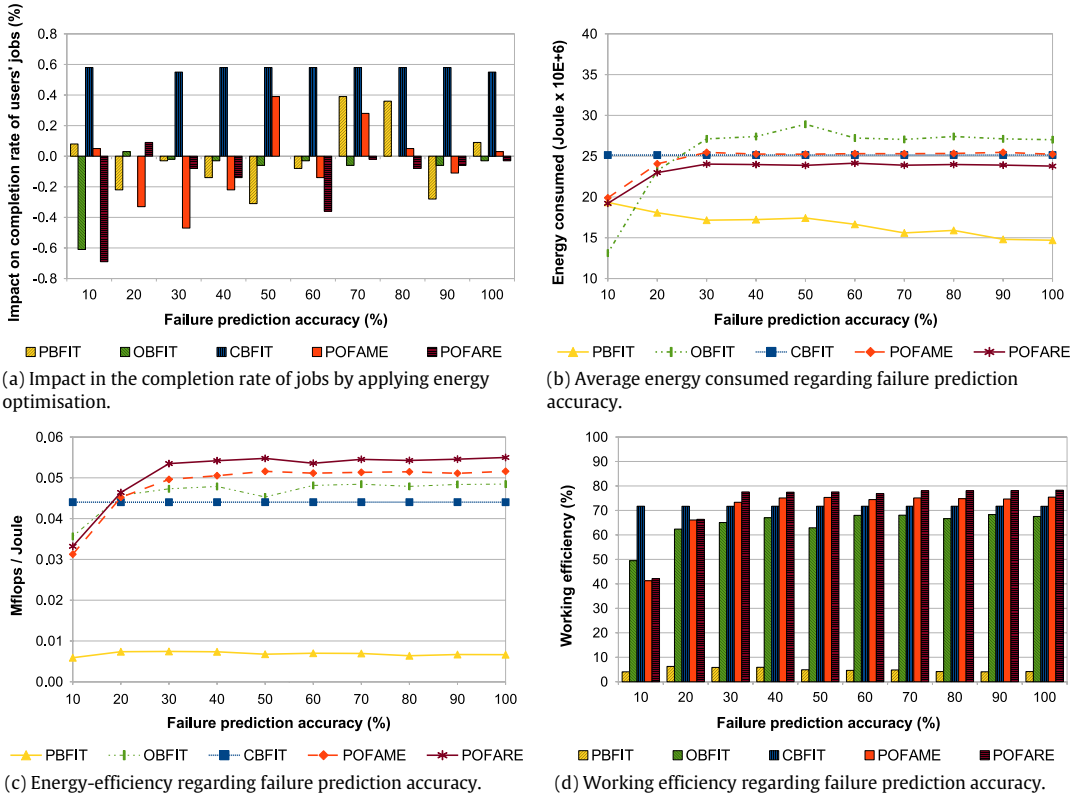
A comparison of the performance results obtained by CBFIT with those obtained by POFAME and POFARE shows that the overhead imposed by pro-active migrations resulted in higher rates of completed jobs, power, energy, and working efficiencies and, therefore, increased the system availability. The number of migrations, as a percentage of the total number of tasks, for the ratios of 0.01 and 0.1 are 0.1% and 3.1% for POFAME and 0.1% and 3.8% for POFARE, respectively. For the ratios of 1 and 10, the number of migrations increases to 126% and 350% for POFAME and POFARE, respectively, which is expected because the tasks are longer than the MTBF. However, POFAME and POFARE still complete more jobs with better efficiency than OBFIT, CBFIT, and PBFIT. The average number of VMs per node for POFARE was 5, 9, 11, and 11 for the ratios of 0.01, 0.1, 1, and 10, respectively.

5.2. Workloads based on Google cloud tracelogs

Fig. 3 shows the results for the workloads introduced in Section 4.2.2, obtained without the dynamic consolidation mechanism. The characteristics of these workloads are well defined with respect to job duration. The goal is to evaluate the performance of the algorithm for a realistic workload. The previously used metric, the average task length to MTBF ratio, is not applicable; thus, we consider the influence of the failure prediction accuracy (FPA) on the algorithm performance [7,22]. FPA is defined as the difference between the actual and predicted failure time, expressed as a percentage. For example, if a physical node is supposed to fail at minute 100, the failure predictor tool will predict the failure at minute 10 if the FPA is 10%. This means that failure prediction inaccuracies have a direct impact on the MTBF perceived by the algorithms that consider node reliability. As before, the MTBF was set to 200 min. If the average task length is 3 min, the average task length to MTBF ratio is 0.015 in this experiment.

The results of CBFIT are constant for all values of FPA, as it does not take failures into account when scheduling and does not perform task migration to prevent failures. When a node fails, CBFIT re-initiates the task in another node. Fig. 3(a) plots the completion rate of users' jobs as the FPA varies from 10% to 100%. We can conclude that, for an FPA below or equal to 20%, a simple best-fit algorithm (CBFIT) completes more jobs than any of the other algorithms. In this case, the failure prediction error generates excessive task migrations than required, thus affecting the rate of completed jobs. Additionally, the energy consumed by OBFIT, POFAME, and POFARE (Fig. 3(b)) is lower for an FPA below or equal to 20% because when a job cannot be finished, the algorithms do not launch their tasks and, therefore, do not spend energy computing that part of the job. The working efficiency (Fig. 3(d)), which combines the rate of jobs completed with the average power efficiency, provides a better evaluation of the algorithm performance. Additionally, CBFIT is only best for an FPA of 10%.

For an FPA greater than or equal to 30%, OBFIT, POFAME, and POFARE achieve a job completion rate near 100%. This result is in accordance with the results of Fig. 2(a) for an average task length to MTBF ratio of 0.01 (in this case, it is 0.015) and an FPA of 75%, where a job completion rate close to 100% was obtained. An important characteristic of POFAME and POFARE is that they achieve an almost constant energy consumption (Fig. 3(b)), energy efficiency (Fig. 3(c)), and working efficiency (Fig. 3(d)), indicating that, for the considered workload, the performance and the system availability are nearly independent of the failure prediction accuracy.



(a) Impact in the completion rate of jobs by applying energy optimisation.

(b) Average energy consumed regarding failure prediction accuracy.

(c) Energy-efficiency regarding failure prediction accuracy.

(d) Working efficiency regarding failure prediction accuracy.

Fig. 4. Energy- and failure-aware scheduling algorithms, with dynamic consolidation, for Google-based workloads. The configuration is: (i) average task length = 3 min; (ii) MTBF = 200 min; (iii) $\tau = 55\%$; (iv) $\gamma = 3$.

PBFIT provides the worse completion rate of users' jobs because most of the tasks are short, which implies that they would be scheduled only if the physical nodes were failing all the time. Consequently, tasks are not scheduled and the energy consumed is also lower than that of the remaining algorithms.

The energy consumed to run the users' jobs vs. the variation in FPA is depicted in Fig. 3(b). Disregarding the PBFIT case that is limited to very few PMs, the results show that POFARE is generally the best algorithm to produce work at lower energy consumption for different values of FPA. The results in Fig. 3(c) show that the POFARE strategy uses the energy more efficiently to produce work with respect to Mflops/Joule, with an improvement of 6.7% compared with OBFIT for an FPA of 75%. The results for the working efficiency in Fig. 3(d) show that for 75% FPA, POFARE yields a 4.8% improvement over OBFIT.

We can conclude that POFARE is the strategy that produces more work, in terms of useful Mflops and number of completed jobs, for less energy. Specifically, if we take into account the fact that the average FPA achieved by the failure predictor tool is approximately 75%, the POFARE algorithm yields 0.049 Mflops/Joule and a working efficiency of almost 68%. These results show that POFARE is the most suitable algorithm to allocate resources to execute cloud users' jobs in an energy-efficient manner.

In the next set of simulations, we assess the performance of the algorithms when using the energy-optimising mechanism to dynamically readjust virtual clusters. With the consolidation mechanism (Algorithm 1, line 6), a VM can be migrated if it benefits the power efficiency.

Fig. 4(a) depicts the impact on job completion rate compared with the results obtained without applying the consolidation mechanism. As before, the impact on POFARE is higher only for an FPA of 10%. For the remaining cases, the reduction in job completion rate is below 0.4% for POFARE. The CBFIT algorithm always improves the job completion rate associated with the power-driven migrations, namely, an improvement of approximately 0.6%.

Fig. 4(b) shows the energy consumed, and, compared with Fig. 3(b), the results show that POFAME, POFARE, and CBFIT reduce the consumed energy when power-oriented migrations are performed. Considering an FPA of 75% and POFARE, the energy consumption, with consolidation, is reduced by almost 11.3% without significantly affecting the job completion rate, which was reduced by less than 0.08%.

Considering Fig. 4(c) and (d), the results show that POFAME and POFARE are clearly the strategies that benefit most from the energy-optimising mechanism. Moreover, POFARE is the algorithm that utilises energy more efficiently to produce work and, for the specific FPA of 75%, yields an improvement over OBFIT of approximately 12.9% and 15.9% with respect to Mflops/Joule and working efficiency, respectively. We can conclude that with consolidation, the improvements of POFARE over OBFIT are significantly greater than those without consolidation.

To analyse the impact of the power-oriented migrations, Table 2 shows results for the Google cloud tracelogs and an FPA of 75%. The consolidation mechanism reduces the energy consumed using POFARE from 26.9 to 23.9 MJ, a reduction of 11.2%, while keeping the same job completion rate. The number of migrations increases from 0.8%, due to failure tolerance migrations, to 2.55%, which also includes power-oriented migrations. OBFIT yields a job completion rate of 100% but with an increase of 13.8% in consumed energy compared with POFARE. The 3.12% of VM migrations for CBFIT, without consolidation, are re-initiates. With consolidation, there is a 1.6% migration rate. From the energy and job completion rate columns in Table 2, we can conclude that the consolidation mechanism improved all algorithms in these two aspects. The average number of VMs per PM is similar for CBFIT, OBFIT, and POFAME. The POFARE algorithm is more relaxed in terms of the power assigned to tasks and, therefore, results in a higher number of VMs per PM. Considering that a computational node may have 64 GB of memory and that each VM requests up to 1 GB of memory, it is feasible to host 21 VMs per node on modern cluster nodes.

Table 2
Results for Google cloud tracelogs for energy, job completion rate, ratio of VM migrations to total number of tasks, and number of VMs per PM for the case of 75% of failure prediction accuracy without consolidation (wocs) and with consolidation (wcs).

Algorithm	Energy (10^6 J) wocs/wcs	Job comp. rate wocs/wcs	VM migrations wocs/wcs	Av. # of VM p/PM wocs/wcs
CBFIT	27.7/25.1	95%/95.5%	(3.12%) + 1.6%	16/17
OBFIT	28.3/27.2	100%/100%	0%/0.99%	17/16
POFAME	28.1/25.3	99.6%/99.8%	1.05%/2.95%	17/17
POFARE	26.9/23.9	99.8%/99.8%	0.8%/2.55%	21/22

Table 3
Results for Google cloud tracelogs without failures for energy, job completion rate, ratio of VM migrations to total number of tasks, and number of VMs per PM, without consolidation (wocs) and with consolidation (wcs).

Algorithm	Energy (10^6 J) wocs/wcs	Job comp. rate wocs/wcs	VM migrations wocs/wcs	Av. # of VM p/PM wocs/wcs
CBFIT	27.8/25.2	100%/99.8%	0%/2.1%	16/17
OBFIT	27.7/25.2	100%/99.9%	0%/1.99%	16/17
POFAME	28.6/25.3	100%/99.8%	0%/2.43%	16/17
POFARE	26.9/23.9	100%/99.9%	0%/1.82%	21/22

The average number of PMs used, for all algorithms, ranges from 7 to 9.

Table 3 shows results for the case where there are no node failures during the execution. We can conclude that POFARE is still the best algorithm, consuming less power for the same rate of completed jobs, with and without consolidation.

5.3. Real platform evaluation

The results presented above are based on a simulation used to evaluate the algorithms for a large set of workloads and failure rates, since it is found to not be feasible or practical, to guarantee repeatable conditions for such a set of experiments [6]. The experiment reported in this section provides preliminary results of the adoption of the proposed algorithms in a real platform. Cloud systems use middle-wares such as OpenStack, Eucalyptus, and OpenNebula, among others. In the current versions, these cloud middle-wares do not allow the vertical scaling of a task, i.e., a task cannot change the percentage of CPU power assigned to it during run time. The algorithm proposed in this paper, POFARE, is based on vertical scaling, i.e., higher or lower, of the CPU power assigned to each task; therefore, we develop a prototype of our Cloud Manager algorithm (Algorithm 1) to perform the real platform evaluation. The experimental testbed consisted of 24 nodes (cores) from an IBM cluster with the following characteristics: each physical server is equipped with an Intel Xeon CPU E5504 composed of 8 cores working at 2.00 GHz, supporting virtualisation extensions and deploying 24 GB of RAM. The Xen 4.1 virtualisation software was installed in the Ubuntu Server 12.04 operating system. We have applied the conditions reported in Section 5.2, with an MTBF of 200 min and an average task length to MTBF ratio of 0.015. Each node behaves based on the description in Section 4.2.3. The cloud manager is made aware of the current status of each node by collecting heartbeat messages [40]. When a node fails, it stops sending heartbeat messages, and the controller subsequently considers that a failure as occurred. The task workload follows the Google cloud tracelogs, as explained in Section 4.2.2. To implement such a workload, the *stress* software (available on Ubuntu Linux), which emulates a workload, is used. We created 151 jobs with a total of 389 tasks that ran for approximately one hour to complete an instance of the experiment. All the characteristics of the jobs, i.e., those used in simulations, are maintained.

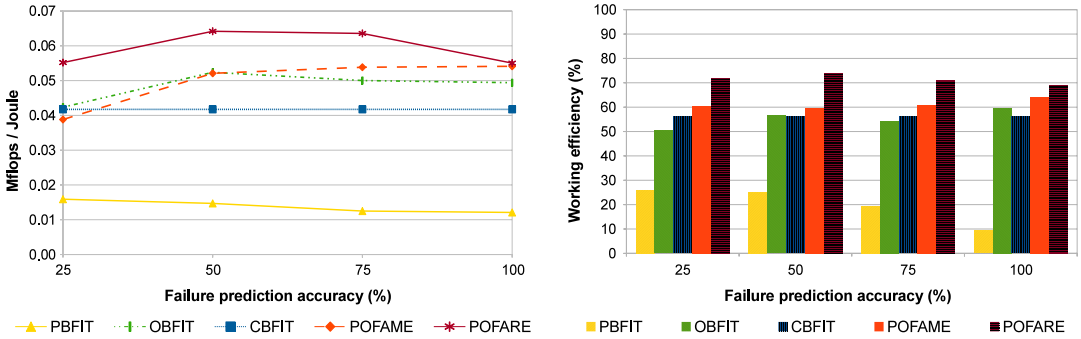
Fig. 5 shows the results obtained from the experiments when applying both energy-optimising and fault-tolerant mechanisms. These experimental results confirm, in general, those obtained through simulations (Fig. 4(c) and (d)). Fig. 5(a) shows that the POFARE algorithm utilises energy more efficiently in terms of Mflops/Joule, outperforming POFAME and OBFIT by 16.4% and by 18.4% on average, respectively. For an FPA of 100%, the delivered Mflops/Joule of the POFARE algorithm decreases to a value near POFAME. This behaviour is justified by the fluctuation of the assigned amount of CPU to each task, which is not constant in the real system, as considered in the simulation. For lower values of FPA, this effect does not affect POFARE because the error margin used to avoid failures accommodates such fluctuations. For FPA values lower than 20%, simple algorithms, such as CBFIT, perform better than any of the other algorithms.

In Fig. 5(b), the results for the working efficiency also demonstrate that for different values of FPA, POFARE obtains a higher degree of consolidation than do all of the other algorithms. POFARE obtains on average 15.2% and 23.5% higher working efficiency than do POFAME and OBFIT, respectively.

In conclusion, the real experiment showed that POFARE consumes less energy than any of the other strategies and that completes a similar number of jobs by their deadlines for FPA values equal to or greater than 25%.

6. Conclusions and future work

In this paper, we have proposed two algorithms that apply proactive fault tolerance to address node failures. The objective is to maximise the useful work performed by the consumed energy in cases where the infrastructure nodes are subject to failure. This objective implies an increase in the amount of useful Mflops processed per energy unit, as well as the number of jobs completed by the consumed power. To achieve this objective, we developed two dynamic VM allocation algorithms, POFAME and POFARE, which use two different methods to provide energy-efficient virtual clusters to execute tasks within their deadlines. While the POFAME algorithm tries to reserve the maximum required resources to execute tasks, POFARE leverages the cap parameter from the Xen credit scheduler to execute tasks with the minimum required resources. The tests were conducted by injecting two sets of synthetic jobs. The first set was generated based on the Poisson distribution and allowed us to assess the scheduling algorithm performance using different average task length to MTBF ratios. The improvement in energy efficiency of POFARE over OBFIT is 23.6%, 16.9%, and 72.4% for the average task length ratios of 0.01, 0.1, and 1, respectively. The improvement in working efficiency of POFARE over OBFIT is 26.2%, 20.3%, and 219.7% for the ratios of 0.01, 0.1, and 1, respectively. The characteristics of the second set of workloads



(a) Energy-efficiency regarding failure prediction accuracy.

(b) Working efficiency regarding failure prediction accuracy.

Fig. 5. Experiments for Energy- and failure-aware scheduling algorithms, with dynamic consolidation, for Google based workloads. The configuration is: (i) average task length = 3 min; (ii) MTBF = 200 min; (iii) $\tau = 55\%$; (iv) $\gamma = 3$.

used the latest version of the Google cloud tracelogs. The results showed that the solutions obtained with the POFARE algorithm are the best, compared with well-known algorithms, such as the best-fit OBFIT algorithm. The results show that the POFARE algorithm improves the work per Joule ratio by approximately 12.9% and the working efficiency by 15.9% compared with the OBFIT results obtained with dynamic optimisation and maintenance of similar levels of completed jobs. The results also showed that a relaxed strategy (POFARE), which assigns the minimum required resources to each task, yields better results than a strategy (POFAME) that assigns the maximum required resources to each task. The preliminary results for a real platform evaluation confirmed the simulation results. Future research will address the platform heterogeneity in both node computation power and energy consumption. With respect to job characterisation, we will also consider data-intensive jobs to increase the range of applicability of the proposed algorithms. Another relevant problem consists of processing workflows in the cloud, such that jobs where tasks may have precedence will also be considered.

Acknowledgement

The authors would like to thank the IBM Portugal Center for Advanced Studies for providing access to a high-performance IBM Cluster, where the real platform experiments were performed.

References

[1] M. Armbrust, A. Fox, R. Griffith, A.D. Joseph, R. Katz, A. Konwinski, et al., Above the clouds: a Berkeley view of cloud computing, Technical Report UCB/EECS 28, Dept. Electrical Eng. and Comput. Sciences, University of California, Berkeley, 2009.

[2] M. Armbrust, I. Stoica, M. Zaharia, A. Fox, R. Griffith, A.D. Joseph, et al., A view of cloud computing, Commun. ACM 53 (4) (2010) 50–58.

[3] G. Vallee, T. Naughton, C. Engelmann, H. Ong, S.L. Scott, System-level virtualization for high performance computing, in: Proceedings of the 16th Euromicro Conference on Parallel, Distributed and Network-Based Processing, PDP 2008, Toulouse, France, 2008, pp. 636–643. http://dx.doi.org/10.1109/PDP.2008.85.

[4] K. Ye, X. Jiang, S. Chen, D. Huang, B. Wang, Analyzing and modeling the performance in Xen-based virtual cluster environment, in: Proceedings of the 12th IEEE International Conference on High Performance Computing and Communications, HPCC 2010, Melbourne, Australia, 2010, pp. 273–280. http://dx.doi.org/10.1109/HPCC.2010.79.

[5] D. Meisner, B.T. Gold, T.F. Wenisch, PowerNap: eliminating server idle power, in: Proceeding of the 14th International Conference on Architectural Support for Programming Languages and Operating Systems, 2009, Washington, DC, USA, pp. 205–216.

[6] A. Beloglazov, J. Abawajy, R. Buyya, Energy-aware resource allocation for efficient management of data centers for Cloud computing, Future Gener. Comput. Syst. 28 (5) (2012) 755–768.

[7] S. Fu, Failure-aware resource management for high-availability computing clusters with distributed virtual machines, J. Parallel Distrib. Comput. 70 (4) (2010) 384–393. http://dx.doi.org/10.1016/j.jpdc.2010.01.002.

[8] A. Sampaio, J. Barbosa, Dynamic power- and failure-aware cloud resources allocation for sets of independent tasks, in: Proceedings of the IEEE International Conference on Cloud Engineering, IC2E 2013, San Francisco, CA, USA, 2013, pp. 1–10. http://dx.doi.org/10.1109/IC2E.2013.16.

[9] Google cluster data V2. URL: http://code.google.com/p/googleclusterdata/wiki/ClusterData2011_1.

[10] J. Xu, J.A.B. Fortes, Multi-objective virtual machine placement in virtualized data center environments, in: Proceedings of the 2010 IEEE/ACM Int'l Conference on Green Computing and Communications & Int'l Conference on Cyber, Physical and Social Computing, GREENCOM'10, Hangzhou, China, 2010, pp. 179–188. http://dx.doi.org/10.1109/GreenCom-CPSCom.2010.137.

[11] J. Xu, J. Fortes, A multi-objective approach to virtual machine management in datacenters, in: Proceedings of the 8th ACM International Conference on Autonomic Computing, ICAC 2011, Karlsruhe, Germany, 2011, pp. 225–234. http://dx.doi.org/10.1145/1998582.1998636.

[12] M.Y. Lim, F. Rawson, T. Bletsch, V.W. Freeh, PADD: power aware domain distribution, in: Proceedings of the 29th IEEE International Conference on Distributed Computing Systems, ICDCS 2009, Montreal, Québec, Canada, 2009, pp. 239–247. http://dx.doi.org/10.1109/ICDCS.2009.47.

[13] C.Y. Lee, A.Y. Zomaya, Energy efficient utilization of resources in cloud computing systems, J. Supercomput. 60 (2) (2012) 268–280.

[14] R. Al-Omari, A.K. Somani, G. Manimaran, An adaptive scheme for fault-tolerant scheduling of soft real-time tasks in multiprocessor systems, J. Parallel Distrib. Comput. 65 (2005) 595–608.

[15] X. Zhu, X. Qin, M. Qiu, QoS-aware fault-tolerant scheduling for real-time tasks on heterogeneous clusters, IEEE Trans. Parallel Distrib. Syst. 60 (6) (2011) 800–812.

[16] E. Feller, L. Rilling, C. Morin, R. Lottiaux, D. Leprince, Snooze: a scalable, fault-tolerant and distributed consolidation manager for large-scale clusters, in: Proceedings of the 2010 IEEE/ACM Int'l Conference on Green Computing and Communications & Int'l Conference on Cyber, Physical and Social Computing, GREENCOM'10, Hangzhou, China, 2010, pp. 125–132. http://dx.doi.org/10.1109/GreenCom-CPSCom.2010.62.

[17] S. Loveland, E.M. Dow, F. LeFevre, D. Beyer, P.F. Chan, Leveraging virtualization to optimize high-availability system configurations, IBM Syst. J. 47 (4) (2008) 591–604. http://dx.doi.org/10.1147/SJ.2008.5386515.

[18] A.B. Nagarajan, F. Mueller, C. Engelmann, S.L. Scott, Proactive fault tolerance for HPC with Xen virtualization, in: Proceedings of the 21st Annual International Conference on Supercomputing, ICS'07, Seattle, WA, USA, 2007, pp. 23–32. http://dx.doi.org/10.1145/1274971.1274978.

[19] L. Ramakrishnan, D. Nurmi, A. Mandal, C. Koelbel, D. Gannon, T.M. Huang, et al., VGrADS: enabling eScience workflows on grids and clouds with fault tolerance, in: Proceedings of the Conference on High Performance Computing Networking, Storage and Analysis, SC'09, Portland, OR, USA, 2009, p. 47. http://dx.doi.org/10.1145/1654059.1654107.

[20] A. Berl, E. Gelenbe, M. Di Girolamo, G. Giuliani, H. De Meer, M.Q. Dang, K. Pentikousis, Energy-efficient cloud computing, Comput. J. 53 (7) (2009) 1045–1051. http://dx.doi.org/10.1093/comjnl/bxp080.

[21] S. Govindan, J. Liu, A. Kansal, A. Sivasubramaniam, Cuanta: quantifying effects of shared on-chip resource interference for consolidated virtual machines, in: Proceedings of the 2nd ACM Symposium on Cloud Computing, 2011, pp. 1–14.

[22] S. Fu, C.Z. Xu, Exploring event correlation for failure prediction in coalitions of clusters, in: Proceedings of the 2007 ACM/IEEE Conference on Supercomputing, SC'07, Reno, NV, USA, 2007, pp. 41:1–41:12. http://dx.doi.org/10.1145/1362622.1362678.

[23] S. Nedevschi, L. Popa, G. Iannaccone, S. Ratnasamy, D. Wetherall, Reducing network energy consumption via sleeping and rate-adaptation, in: Proceedings of the 5th USENIX NSDI, 2008, pp. 323–336.

[24] W. Kim, M.S. Gupta, G.-Y. Wei, D. Brook, System level analysis of fast, per-core DVFS using on-chip switching regulators, in: IEEE 14th International Symposium on High Performance Computer Architecture, HPCA, 2008, pp. 123–134.

[25] S. Fu, Failure-aware construction and reconfiguration of distributed virtual machines for high availability computing, in: 9th IEEE/ACM International Symposium on Cluster Computing and the Grid, 2009, pp. 372–379.

[26] C. Clark, K. Fraser, S. Hand, J.G. Hansen, E. Jul, C. Limpach, I. Pratt, A. Warfield, Live migration of virtual machines, in: Proceedings of the 2nd Symposium on Networked Systems Design and Implementation, NSDI 2005, USENIX, Boston, USA, 2005.
[27] M.R. Hines, K. Gopalan, Post-copy based live virtual machine migration using adaptive pre-paging and dynamic self-ballooning, in: Proceedings of the ACM International Conference on Virtual Execution Environments, 2009, pp. 51–60.
[28] A. Sampaio, J. Barbosa, Optimizing energy-efficiency in high-available scientific cloud environments, in: Proceedings of the IEEE International Conference on Cloud and Green Computing, CGC 2013, Karlsruhe, Germany, 2013, pp. 76–83.
[29] M. Stillwell, F. Vivien, H. Casanova, Dynamic fractional resource scheduling versus batch scheduling, IEEE Trans. Parallel Distrib. Syst. 23 (3) (2012) 521–529. http://dx.doi.org/10.1109/TPDS.2011.183.
[30] P. Barham, B. Dragovic, K. Fraser, S. Hand, T. Harris, A. Ho, R. Neugebauer, I. Pratt, A. Warfield, Xen and the art of virtualization, in: Proc. 19th ACM Symposium on Operating Systems Principles, SOSP 2003.
[31] Z. Shen, S. Subbiah, X. Gu, J. Wilkes, CloudScale: elastic resource scaling for multi-tenant cloud systems, in: Proceedings of the 2nd ACM Symposium on Cloud Computing, 2011.
[32] V. Berten, J. Goossens, E. Jeannot, On the distribution of sequential jobs in random brokering for heterogeneous computational grids, IEEE Trans. Parallel Distrib. Syst. 17 (2) (2006) 113–124.
[33] Z. Liu, S. Cho, Characterizing machines and workloads on a Google cluster, in: Proceedings of the International Conference on Parallel Processing Workshops, ICPPW 2012, Pittsburgh, PA, USA, 2012, pp. 397–403. http://dx.doi.org/10.1109/ICPPW.2012.57.
[34] C. Reiss, A. Tumanov, G.R. Ganger, R.H. Katz, M.A. Kozuch, Towards understanding heterogeneous clouds at scale: Google trace analysis, Intel Science and Technology Center for Cloud Computing, Technical Report, 2012, URL: http://www.pdl.cs.cmu.edu/PDL-FTP/CloudComputing/ISTC-CC-TR-12-101.pdf.
[35] C. Walck, Hand-book on statistical distributions for experimentalists, Internal Report SUF-PFY/96-01, Particle Physics Group, Fysikum University of Stockholm, Stockholm, 2007.
[36] I.S. Moreno, P. Garraghan, P. Townend, J. Xu, An approach for characterizing workloads in Google cloud to derive realistic resource utilization models, in: Proceedings of the IEEE International Symposium on Service-Oriented System Engineering, SOSE 2013, San Francisco, CA, USA, 2013, pp. 49–60. http://dx.doi.org/10.1109/SOSE.2013.24.
[37] B. Schroeder, G.A. Gibson, A large-scale study of failures in high-performance computing systems, IEEE Trans. Dependable Secure Comput. 7 (4) (2010) 337–350. http://dx.doi.org/10.1109/TDSC.2009.4.
[38] H.A. Lagar-Cavilla, J.A. Whitney, A. Scannell, P. Patchin, S.M. Rumble, Eyal de Lara, M. Brudno, M. Satyanarayanan, SnowFlock: rapid virtual machine cloning for cloud computing, in: Proceedings of 4th ACM European Conference on Computer Systems, EuroSys, 2009, pp. 1–12.
[39] Linpack, 2012. URL: http://www.netlib.org/linpack/.
[40] W. Zhao, P.M. Melliar-Smith, L.E. Moser, Fault tolerance middleware for cloud computing, in: IEEE 3rd International Conference on Cloud Computing, July 2010, pp. 67–74.
[41] L. Cherkasova, D. Gupta, A. Vahdat, Comparison of the three CPU schedulers in Xen, Perform. Eval. Rev. 35 (2) (2007) 42.

Altino M. Sampaio received his B.Sc. degree in Electrical and Computer Engineering from the University of Porto, Porto, Portugal, in 2002. Since 2008 he serves as a teacher assistant in several computer science courses at Instituto Politécnico do Porto, Porto, Portugal. He started his Ph.D. studies in computer science at the University of Porto, Portugal, in 2009. His research interests are in the fields of Cloud and distributed computing systems, especially resources scheduling, considering fault tolerance and energy optimisation in Cloud Computing.

Jorge G. Barbosa received the B.Sc. degree in Electrical and Computer Engineering from the Faculty of Engineering of the University of Porto (FEUP), Portugal, the M.Sc. degree in Digital Systems from the University of Manchester Institute of Science and Technology, England, in 1993, and the Ph.D. degree in Electrical and Computer Engineering from FEUP, Portugal, in 2001. Since 2001 he is an Assistant Professor at FEUP. His research interests are related to parallel and distributed computing, heterogeneous computing, scheduling in heterogeneous environments, cloud computing and biomedical engineering.