2014 IEEE 6th International Conference on Cloud Computing Technology and Science

Energy-Efficient Scheduling of Urgent Bag-of-Tasks Applications in Clouds through DVFS

Rodrigo N. Calheiros and Rajkumar Buyya
Cloud Computing and **D**istributed **S**ystems (CLOUDS) Laboratory
Department of Computing and Information Systems
The University of Melbourne, Australia
Email: {rnc, rbuyya}@unimelb.edu.au

***Abstract*—The broad adoption of cloud services led to an increasing concentration of servers in a few data centers. Reports estimate the energy consumptions of these data centers to be between 1.1% and 1.5% of the worldwide electricity consumption. This extensive energy consumption precludes massive CO_2 emissions, as a significant number of data centers are backed by "brown" power plants. While most researchers have focused on reducing energy consumption of cloud data centers via server consolidation, we propose an approach for reducing the power required to execute urgent, CPU-intensive Bag-of-Tasks applications on cloud infrastructures. It exploits intelligent scheduling combined with the Dynamic Voltage and Frequency Scaling (DVFS) capability of modern CPU processors to keep the CPU operating at the minimum voltage level (and consequently minimum frequency and power consumption) that enables the application to complete before a user-defined deadline. Experiments demonstrate that our approach reduces energy consumption with the extra feature of not requiring virtual machines to have knowledge about its underlying physical infrastructure, which is an assumption of previous works.**

I. Introduction

In the last years, we witnessed a shift in focus from traditional IT models to cloud computing [1]. Cloud computing enables IT services to be offered to customers as utilities: physical resources hosting the application are obtained from a resource pool and accessed via the Internet. The amount of resources allocated to a service can be dynamically modified to better supply its demand (a feature called *elasticity*), and the utilization of resources is metered. Regarding the particular IT service model offered by the provider, they broadly can be classified as Infrastructure as a Service (IaaS), where customers lease virtual machines (VMs) and other low-level services, such as storage; Platform as a Service (PaaS), where customers acquire platforms and frameworks for hosting applications; and Software as a Service (SaaS), where users use applications hosted in the data center rather than in their local systems. This work focuses on IaaS clouds.

The recent surge of cloud service providers led to an increasing concentration of IT servers in a few data centers. A point of increasing concern is the amount of energy consumed by each of these data centers. Reports estimate the energy consumptions of data centers to be between 1.1% and 1.5% of the worldwide electricity consumption [2]. This massive energy consumption causes significant CO_2 emissions, as many data centers are backed by "brown" powerplants.

State-of-the-art research in energy-efficient cloud computing focuses on the problems of VM consolidation and selection of energy sources for data centers (see Section II). In this paper, we consider a different approach, orthogonal to these studies, which aims at reducing the power required to execute urgent, CPU-intensive Bag of Tasks (BoT) applications on cloud infrastructures. A urgent application is a High Performance Computing application whose execution needs to complete before a user-defined deadline because of its utilization in sensitive contexts, such as disaster management and healthcare [3]. We define a deadline-constrained application as an application that needs to have its execution completed before a user-defined *soft deadline*. Notice that, in contrast to a hard deadline, a soft deadline does not render the computation useless if the deadline is violated. Instead, there exists a utility value associated to the computation whose value is maximized if the application completes by the deadline, and is increasingly reduced as the completion is delayed [4].

In this direction, the contribution of this paper is the proposal of an algorithm that exploits intelligent scheduling combined with the Dynamic Voltage and Frequency Scaling (DVFS) capability of modern CPU processors to keep the CPU operating at the minimum voltage level (and consequently minimum frequency and power consumption) that enables the application to complete before a user-defined deadline. The proposed algorithm is more energy-efficient than existing approaches for the problem, that are not customized for cloud environments and therefore are unable to make the best decision when applied in virtualized environments.

Our approach applies DVFS at the middleware/Operating System level rather than at CPU level. This means that maximum frequency levels to be assigned during tasks execution are supplied by our algorithm. If our approach is combined with DVFS techniques applied at the CPU level to manage frequency for operation at the microinstruction level [5], [6], even more savings could be expected. Experiments are presented that demonstrate that our algorithm significantly reduces energy consumption with the extra feature of not requiring virtual machines to have knowledge about its underlying physical host, which is an assumption of previous solutions for the problem.

978-1-4799-4093-6/14 $31.00 © 2014 IEEE
DOI 10.1109/CloudCom.2014.20

II. Related Work

To counter the increasing energy consumption and greenhouse gases emissions caused by cloud data centers, recent studies focused on optimizing the utilization of hosts via dynamic consolidation (with or without reconfiguration) of virtual machines [7]–[10]. In this model, VMs with low utilization are placed together on a single host, which enables the other hosts initially hosting the VMs to be shut down. Laszewski et al. [11] developed an algorithm for energy-efficient scheduling of VMs in hosts that are part of a virtualized cluster. These approaches see VMs as "black boxes" and thus cannot guarantee the deadlines of urgent applications within the VM.

Li et al. [12] developed an energy-aware algorithm for scheduling tasks on heterogeneous clusters. However, the approach tries to minimize the execution time without enforcing deadline for applications. It also does not explore DVFS. Rather, it takes into account the different power consumption incurred by heterogeneous machines.

Chetsa et al. [13] developed a technique where characteristics of HPC applications are inferred at runtime and measures for energy savings are applied based on the characteristics of the application. Although the approach does not address the problem of deadline-constrained applications, it can be used in conjunction with our algorithm. This is because we focus on dynamic scaling the CPU frequency, whereas Chetsa et al. approach also applies energy-saving measures to other system components, such as network, hard disk, and memory.

The utilization of DVFS as a means to reduce energy consumption of applications has been explored in the last years in related areas. Ruan et al. [14] and Wang et al. [15] applied the technique in workflow applications on clusters, whereas Tian et al. [16] apply the approach for web servers. Our approach is designed for urgent, CPU-intensive BoT applications, where there are no execution dependencies between tasks, and works based on user-defined deadlines for application execution.

Eyerman and Eeckhout [5] and March et al. [6] propose techniques for fine grain control of frequency and voltage at the CPU level. Eyerman and Eeckhout [5] explores regulation of frequency to slow down the flow of CPU instructions through CPU units (such as buffers and functional units) in the event of cache misses that disrupt the flow of execution of operations being processed. March et al. [6] apply DVFS and task migrations to balance load among CPU cores, what in some circumstances enables the whole CPU to execute at a lower frequency/voltage. These approaches are complementary to ours, as they operate at the CPU level whereas our approach is applied at middleware/Operating System level. Thus, one could expect further reduction in energy consumption if our approach is applied in conjunction with Eyerman and Eeckhout's approach at CPU level.

Kim et al. [17] and Zhang et al. [18] developed heuristics for scheduling deadline-constrained BoT applications in clusters and heterogeneous distributed systems, respectively. Kim et al. [17] approach requires each execution node to have knowledge about the total consumption from the underlying hardware. Although this is a reasonable approach for clusters, this is not reasonable in cloud systems, where nodes are virtual machine that, because of security issues, cannot have access to information about energy consumption from other VMs in the same host. Zhang et al. [18] approach is also not tailored for cloud environments, where VMs share the same hardware and therefore their potential co-location affects the total consumption caused by the host. Therefore, a different approach, like ours, is necessary for cloud infrastructures.

III. System and Application Models

Our target system model is depicted in Figure 1. It is composed of a virtualized IaaS cloud data center that supports a PaaS layer that is available to multiple users for execution of urgent, CPU-intensive BoT applications. The data center is composed of a number of (potentially heterogeneous) physical servers (hosts). Virtual machines are executed in the servers, and they are managed by a virtual machine manager installed on each host. *We also assume that a dedicated amount of servers from the cloud is available for execution of the urgent applications, and therefore it does not suffer from interference of other services or other tasks hosted in the same cloud*. It means that any task already executing in the resources are subject to the proposed approach.

VMs do not share CPU cores with other VMs in the same host. It means that, apart from a small fraction of CPU time reclaimed by the virtual machine manager for its operation, the CPU core is exclusively available for the VM (a configuration that can be obtained in current virtual machine managers such as Xen [19]). Each VM has exclusive use of a share of the host's memory. In order to control the access to VMs by jobs from multiple users, the PaaS layer contains a Resource Management System (RMS) that coordinates the execution of applications submitted by users in the infrastructure. In the context of clouds, a RMS is a key element of the PaaS that is responsible for provisioning and management decisions for resources and incoming requests.

The PaaS layer maintains one queue for each VM to store tasks scheduled for execution until the VM is idle. Only one task is executed simultaneously on each VM, and if more than one task is scheduled to the same VM, the remaining tasks are kept the queue. Formally, we define for the data center a set $VM = \{v_0, v_1, ..., v_n - 1\}$ of n available VMs.

The application model consists of urgent, CPU-intensive Bag-of-Tasks applications submitted by users of the PaaS service. The job execution request for a job j_i contains the following information: (i) the independent tasks $t_0, t_1, ..., t_n$ that compose the job; (ii) estimated runtime $rt_{max}(t_k)$ of each task t_k executed at the maximum frequency of the CPU. This information can be obtained via previous profiling and/or from historical data from previous executions of the job; and (iii) a deadline $dl(j_i)$ for the job to complete. Tasks are non preemptable, and the completion time of the job is computed as the completion time of the last task. It means that the

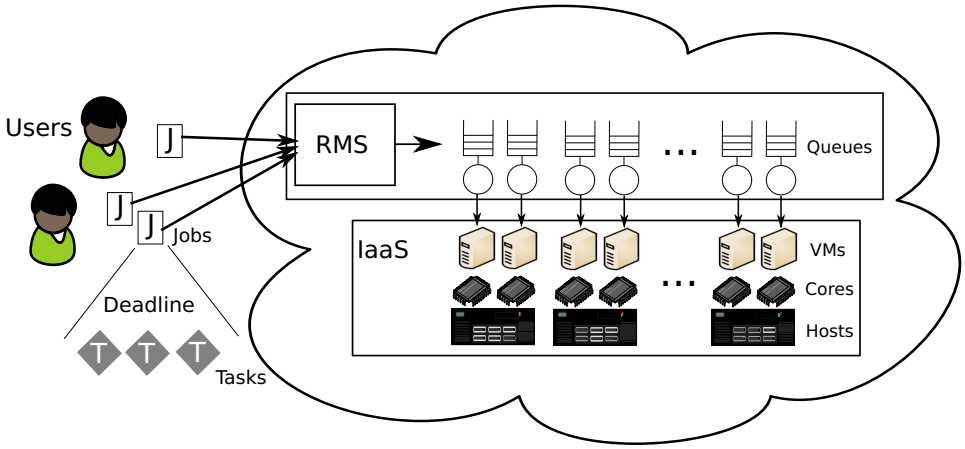

Fig. 1. System and application models adopted in this paper. A cloud data center containing VMs with dedicated CPU cores execute, on its PaaS layer, urgent, CPU-intensive BoT applications (jobs) from multiple users. The CPU frequency of each core/VM can be independently adjusted in order to balance energy utilization and application deadlines.

scheduling algorithm has to operate in such a way that all the tasks that compose the job are finished before the job deadline.

In the proposed model, jobs have "soft deadlines", which means that violations on deadlines do not render the computation irrelevant, but reduces its value for the user [4]. As loss of value of the computation is proportional to the amount of delay for job completion regarding its deadline, it is important that deadlines are not missed at all, or are missed by just a small margin. Deadlines missed by large margins result in the computation being useless for the user and therefore the energy spent on its computation can be considered wasted. Therefore, if the PaaS layer, via its RMS, determines that the job deadline cannot be met, the request is rejected.

The above model is a generalization of a more restrictive model where users submit requests for specific computational-intensive services whose details of the underlying application (e.g., video transcoding, data analytics) is known by the provider and enhanced with the concept of execution soft deadline. An example of a commercial cloud service applying this more restrictive model without deadlines is Amazon Elastic Transcoder[1].

The objective of the cloud platform is to balance the energy consumption of the system and comply with application deadlines. This is achieved with the proper configuration of the frequency of VMs to speed up or slow down the CPU core allocated to each VM (and consequently reduce its energy consumption). We assume that the frequency and voltage of each core can be individually managed (a set up that is demonstrated [20] to provide more energy efficiency than per-chip DVFS) by the RMS. Formally, each host h_l of the data center has r cores, $c_1 ..., c_r$. Therefore, different hosts can have different number of cores. Each core c can be in one of l^c different CPU levels, where l_0^c is the idle state of core c and l_{max}^c is the state of maximum frequency (and energy consumption) of core c. Without loss in generality, we assume that the number of states can be different among cores belonging to different hosts, but are the same for cores in the same host.

As we target CPU-intensive applications, we assume that performance of core c is linearly proportional to the frequency level for frequency levels between l_0^c and l_{max}^c. The power consumed by each host is calculated based on the individual contribution of each core of the host. Furthermore, we assume that hosts that are not in use (i.e., all its VMs are idle, and therefore their respective CPU cores are in the state l_0^c) are suspended until one of the VMs is necessary, so the host is restored to a full operational mode. Also, as the time for scaling the CPU frequency is at the scale of nanoseconds while application execution is minutes or hours [5], we assume as negligible the time taken by the CPU core to reach the desired frequency level. The total energy consumed by a host is a function of the frequency level of each of its cores.

When a new job is received by the system, the RMS decides in which VM the task will be executed, in what position of the queue it will be placed, and the frequency level to be applied to the VM.

IV. Cloud-Aware Scheduling Algorithm

The rationale behind our proposed scheduling algorithm is to dynamically scale the frequency and voltage of CPUs assigned to a virtual machine in such a way that tasks sequentially executed in the VM complete before their deadlines. As discussed in Section II, the idea has been explored in clusters in the past. One key difference between traditional (i.e., non-virtualized) clusters and clouds is that, in the former, calculation of energy consumption incurred by applications is straightforward, as there is no abstraction of physical resources as in the case of clouds.

In a cloud environment, hardware is shared among VMs. It means that the energy consumption of a host is not determined by a single VM, but by the combined state of all the VMs. Furthermore, when scheduling a task, the physical location of

[1]http://aws.amazon.com/elastictranscoder/

the VM can be taken into consideration, in such a way that requests are preferentially submitted to VMs whose host is already in use (i.e., other VMs in the same host are already executing tasks). This enables the system to consolidate the load whenever possible. This in turns helps in reducing the total energy consumption of the infrastructure, as it enables unused hosts to be suspended or kept in low power states. Schedulers unaware of the virtualized system are unable to make decisions based on the location of the VM and therefore cannot make such kind of optimization.

The cloud-aware scheduling algorithm is listed in Algorithm 1. For selecting the most suitable VM for each task, it adopts the concepts of *score* of the selection and *energy ranking* of hosts. For the purpose of this algorithm, the best solution is the one that requires the least extra energy to operate. Therefore, from the best to the worst approach, the possible scheduling decisions are: (i) the task is scheduled to a VM in use, and does not require its frequency to be increased; (ii) the task is scheduled to a VM in use, but the CPU frequency has to be increased; (iii) the task is scheduled to an idle VM, but the host contains at least one VM that is not idle; and (iv) the task is scheduled to an idle VM from an idle host, i.e., the host has to be taken from its low-energy state to execute the task. *Integer constants are applied in the algorithm in order to enforce the above order for host selection.* The only requirement for setting such parameters is that the value assigned to the score of an option is bigger than the maximum value that can be achieved by the score of the next option in the preference order, even in the case of some deductions are applied by the algorithm. If the number of frequency levels is equal or smaller than ten, the base score for each of the four decisions above can be set, *without loss of generality*, as 10000, 1000, 100, and 10 (so the last option never assumes a negative value if it has to go from the lowest frequency to the highest frequency). Any other values could be chosen, as long as subtractions in lines 17, 19, 25, and 27 never result in a negative score value or result in the score assuming the value of the score value of next possible scheduling decision.

The energy ranking is an abstraction adopted by the algorithm for classifying different hosts from the infrastructure. The more energy efficient the host, the higher its rank. In the context of the algorithm, ranks are applied for each particular server type (hosts of the type of the most efficient server is ranked 0, the second most efficient 1, and so on). Alternatively, groups of server types with close energy performance can be grouped within a single rank, so for the purpose of the algorithm, hosts that are of any of the grouped types are considered as having the same efficiency.

For the above concepts to be applied, the algorithm, for each task, iterates over each VM (Lines 6–40). It first verifies the utilization status of the VM. If the VM is in use (Line 10), the task will have to be inserted in the existing non-empty execution queue of the VM. The queue is arranged so that tasks are kept in ascending order of deadline. The new task is temporarily (until the check for the feasibility of the deadline is performed) inserted in the correct position in the list (Line

Algorithm 1 Cloud-Aware Energy-Efficient Scheduling.

1: **for each** task $t_i \in J$ **do**
2: $chosenVm \leftarrow$ null;
3: $chosenPosition \leftarrow$ null;
4: $chosenFrequency \leftarrow$ null;
5: $maxScore \leftarrow 0$;
6: **for each** VM $v_n \in VM$ **do**
7: $score \leftarrow 0$; $position \leftarrow 0$; $freqRank \leftarrow 0$;
8: $energyRank \leftarrow$ energy ranking of the host where the VM runs;
9: **if** there are tasks scheduled to v_n **then**
10: Insert t_i in the scheduling list so that the list is sorted in non-decreasing order of task deadline;
11: $position \leftarrow$ position of t_i in the scheduling list;
12: $freqRank \leftarrow$ index of the smallest frequency level able to meet the deadline of all tasks in the scheduling list, or -1 if no frequency meets all the deadlines;
13: **if** $frequencyRank = -1$ **then**
14: $score = -1$;
15: **else**
16: **if** $freqRank >$current frequency of v_n **then**
17: $score \leftarrow 1000 - freqRank - energyRank$;
18: **else**
19: $score \leftarrow 10000 - freqRank - energyRank$;
20: **end if**
21: **end if**
22: **else**
23: $freqRank \leftarrow$ index of the smallest frequency level able to meet the deadline of t_i, or -1 if no frequency meets the deadline;
24: **if** $freqRank > -1$ and the host where v_n runs contains at least 1 VM not idle **then**
25: $score = 100 - energyRank$;
26: **else if** frequency> -1 **then**
27: $score = 10 - energyRank$;
28: **else**
29: $score = -1$;
30: **end if**
31: Insert t_i in the empty scheduling list for v_n;
32: $position \leftarrow 0$;
33: **end if**
34: **if** $score > maxScore$ **then**
35: $maxScore \leftarrow score$;
36: $chosenPosition \leftarrow position$;
37: $chosenFrequency \leftarrow frequencyRank$;
38: $chosenVm \leftarrow v_n$;
39: **end if**
40: **end for**
41: **if** $chosenVm = null$ **then**
42: Remove all scheduled tasks from J from the scheduling queues;
43: **Return** failure.
44: **end if**
45: **end for**
46: **Return** success.

10). If other task with the same deadline is also in the queue, it has priority over the arriving task.

The next step consists on the computation of the lowest frequency level that can meet the deadline of all tasks (Line 12). If the deadline cannot be met even at the highest frequency, this VM is not suitable for executing the task, and the score of this VM is set to -1 (Line 14). Otherwise, the VM can be used for the task. In this case, the score is determined based on the necessity of increasing the frequency level (Line 17) or not (Line 19). The base value of the score function is set to 1000 or 10000, respectively. Because a similar solution (for example, scheduling without changing the frequency level) can be achieved for different VMs, the algorithm prioritizes the VM that can run in the smaller frequency. This is achieved via subtraction of the frequency level from the base value.

If the VM being iterated is idle, its score is based on the state of the host, as stated previously. If the task cannot have its deadline met even at the highest CPU frequency, the score value is -1, as in the previous case. Otherwise, a score value of 100 is assigned if the host is in use and a value of 10 is assigned if the host is idle (Lines 24–27). The CPU frequency, in this case, is the lowest one able to meet the task deadline.

In any case, the rank of the host is debited from the score, so if two selections have the same score, the algorithm prioritizes the one with the highest rank (and therefore belonging to the more energy-efficient class of hosts). If many ranks are available, the base value of the score function must be increased to avoid a valid solution (i.e., a solution whose deadline of all tasks are met) resulting in a negative score. However, it does not require any change in the dynamics of the algorithm.

The selection of the best VM so far for the task occurs between lines 34 and 39. Because $maxScore$ is initially set to 0, negative score values (i.e., unmet deadlines) do not lead to update of the VM to be chosen. It means that, if no VM can meet the deadline, $chosenVm$ will maintain its initial value $null$ and this is regarded as a failure in scheduling the task, and consequently the job (Lines 41–43). In this case, the scheduled tasks are removed from the queues and the job is rejected. If all tasks can be successfully scheduled, the process is considered successful (Line 46), and the job is accepted for execution.

A. Complexity Analysis

The outer loop of the Algorithm 1 (Lines 1–45) iterates over each task of J. Given n the number of tasks of J, then the complexity incurred by this loop is $O(n)$. The inner loop (Lines 6–40) iterates for each VM, and therefore it is $O(m)$, where m is the number of VMs in the cloud.

The insertion in Line 10 requires, in the worst case, all the elements in the list to be read. The worst case scenario in this case occurs when all tasks are assigned to the same VM, and therefore the size of the list will be the number of scheduled tasks in the VM. As this requires insertion of elements in a sorted sequence, efficient data structures (such as Red-Black trees) can perform each insertion in $O(\log n)$. In total, this operation is repeated for each of the n tasks (the outer loop), and the insertion operation in the worst case will be performed in $O(n \log n)$. In this situation, still all the VMs are tested, although the other cases led to $O(1)$ operations. Therefore, the complexity at this point of analysis is $O(mn \log n)$.

The statements in Lines 12 and 23 require, in the worst case, all the frequencies to be tested, leading to a worst case $O(l)$, where l is the available frequency levels. However, given that $l << mn \log n$, because while VM are expected to be in the order of hundreds or thousands, CPUs are expected to have frequency levels in the order of tens and the tasks are also expected to be in higher number than the number of frequency levels. This allows us to ignore the term in the analysis, as it is dominated by the number of VMs and tasks, to establish the asymptotic complexity of the algorithm as $O(mn \log n)$, where m is the number of VMs and n is the number of tasks.

This complexity is acceptable (i.e., it does not delay too much the execution of the tasks and thus does not compromise the capacity of the system to complete the tasks within their deadlines) given that, typically, the average number of tasks in a BoT application is between 5 and 50 [21], which is much smaller than the expected number of VMs in a typical infrastructure. When the number of tasks or VMs is too large, the scheduling process could be split so different nodes compute the scheduling for a subset of hosts/VMs, and the partial results are gathered and used to decide the final placement of the task.

V. Performance Evaluation

In order to evaluate the effectiveness of our proposed algorithm for energy efficient scheduling of BoT applications in clouds, we performed simulation experiments using the CloudSim toolkit [22]. The simulated environment consists of a cloud data center hosting 200 hosts. Each host has four cores and 8 GB of RAM. Each host supports up to four virtual machines, amounting to 800 VMs in the data center. The energy consumption model assumed for each host is the one observed from an IBM System x3250 M3 [10]. Although our algorithm supports heterogeneous hosts, we limited this experiment to a homogeneous data center in order to enable us to focus on the effects different workloads have on the performance of the algorithm.

The BoT workload that is submitted to the cloud is based on the model proposed by Iosup et al. [21]. The analysis established that the arrival of jobs has different behavior in peak and in off-peak times. Nevertheless, to further stress the infrastructure for the purpose of these experiments, we applied the inter-arrival rate at the peak time for the observed 24-hour period. Such an inter-arrival rate is approximated by a Weibull distribution with parameters (4.25, 7.86) [21].

For each incoming job execution request, the number of tasks that composes the job is given by 2^x, where x is a Weibull distribution with parameters (1.76, 2.11). Execution time of tasks of the same job is assumed to be homogeneous and given by 2^x minutes, where x follows a normal distribution with average 2.73 and standard deviation 6.1.

TABLE I
AVERAGE NUMBER OF JOBS GENERATED BY EACH WORKLOAD, FOR URGENCY RATE OF 20% (URGENCY RATE OF 50% IN PARENTHESIS).

	$\alpha = 4.25$	$\alpha = 8.5$
$\beta = 15.72$	6040.9 (6041.0)	5818.5 (5823.2)
$\beta = 3.93$	24163.2 (24170.0)	23285.9 (23313.8)
$\beta = 7.86$	12099.4 (12085.9)	11636.4 (11637.7)

The traces that originated the model we utilize do not contain information about job deadlines. Thus, we assigned deadlines based on the following method. Jobs are divided in two urgency classes, namely *low-urgency* and *high-urgency* jobs. Jobs are assigned to each class uniformly according to a defined share. We considered in these experiments two different proportions of high-urgency jobs, namely 20% and 50%.

Although all the jobs have a deadline regardless of their urgency class, deadlines of high-urgency and low-urgency jobs have different ratios between the given deadline and the task runtime. High-urgency jobs have such a rate sampled from a uniform distribution with average 3 and standard deviation 1.4 (i.e., in average the deadline is 3 times of the estimated runtime), whereas low-urgency jobs have such a rate sampled from a uniform distribution with average 8 and standard deviation 3. The obtained value for deadline is counted from the moment the job is submitted for execution.

To enable our algorithm to be evaluated under different load conditions, we conducted experiments with varied arrival rates of jobs by modifying the parameters of the Weibull distribution that defines the inter-arrival time. Experiments were repeated using two different values for the scale of the distribution (first parameter of the Weibull distribution): 4.25, as in the traces, and 8.5. We also tested three different values for the shape (second parameter of the Weibull distribution): 7.86, as in the traces, along with 3.93 and 15.72. Table I presents the number of jobs generated by each workload.

The 24 hours-long workloads with different arrival rates were submitted for execution in the simulated cloud. Experiments for each combination of arrival rates (six different combinations of shape and scale) and two rates of urgent requests (20% and 50%), amounting to 12 different workloads, were repeated 10 times. The collected output metrics are:

- **Job execution time:** The time taken from the moment the job is submitted to the moment the last task of the job completes its execution;
- **Rejection rate:** The amount of jobs that were rejected because the algorithms were unable to find a schedule that would allow the job to complete before its deadline;
- **Deadline violations:** Proportion of accepted jobs whose deadline was violated (i.e., the job execution finished after the established deadline);
- **Energy consumption:** Total energy consumption incurred by the servers to execute the workload.

The reported output is the average for the 10 repetitions.

A. Baseline Algorithm

There are a few algorithms available in the literature that apply DVFS to execute applications. As discussed in Section II, some of them target different application models [14], [15], whereas other focus on other environments, such as homogeneous clusters [17] and heterogeneous clusters [18].

In order to provide a compatible baseline for experiment purposes, and to demonstrate the advantages of being aware of the virtualized environment when making energy-efficient scheduling decisions for urgent applications, we use as a baseline a modified version of our proposed algorithm that is not cloud-aware. Such approach, referred to as *baseline* in the results, operate similarly to our approach when reusing VMs. It means that computations related to positioning, frequency selection, and ranking are the same as our cloud-aware algorithm when the loop starting on Line 6 of Algorithm 1 is considering a VM that is already in use. However, when the algorithm is considering a VM that is not in use, it cannot differentiate cases where the host is in use from cases where hosts are not in use. Therefore, the conditional statement on Algorithm 1 Line 24 is ignored, and the same score of $100 - energyRank$ is returned when this point of the algorithm is reached.

The simulation scenario described previously is subject to scheduling and execution through our proposed algorithm and through the baseline algorithm described in this section. Results are presented and discussed next.

B. Results and Discussion

Figure 2 presents the average energy consumption incurred by each algorithm for each workload. Our algorithm performs better than the baseline: energy consumption is between 2% and 29% better than the baseline approach. The graphs show that the workload (in terms of inter-arrival and number of requests) has a big impact on the performance of the algorithm. Furthermore, the difference in the comparative performance between the two urgency rates show that the performance is affected by the urgency of requests: when there are more high urgent requests, the algorithm has little options for optimization and needs to use more machines, and thus the initial energy-efficient placement is outweighed by the need for extra machines to run the urgent requests.

Table II shows the rejection rate of requests for both algorithms. Rejections from both algorithms are the same for most scenarios, and in all the others the cloud-aware algorithm was able to generate a slightly smaller rejection rate. This means that the reduction in energy consumption caused by our algorithm is a result of a more effective scheduling of the workload, and that the savings in energy do not impact the acceptance rate of requests. The table also shows that the main reason for request rejections is unfeasibility of the request rather than poor decisions by the algorithms.

Besides the quantitative advantages of our cloud-aware algorithm demonstrated in our experiments, it also has the advantage that it delegates the estimation of energy consumption to the RMS, whereas existing approaches require each compute node to estimate its own energy consumption, which

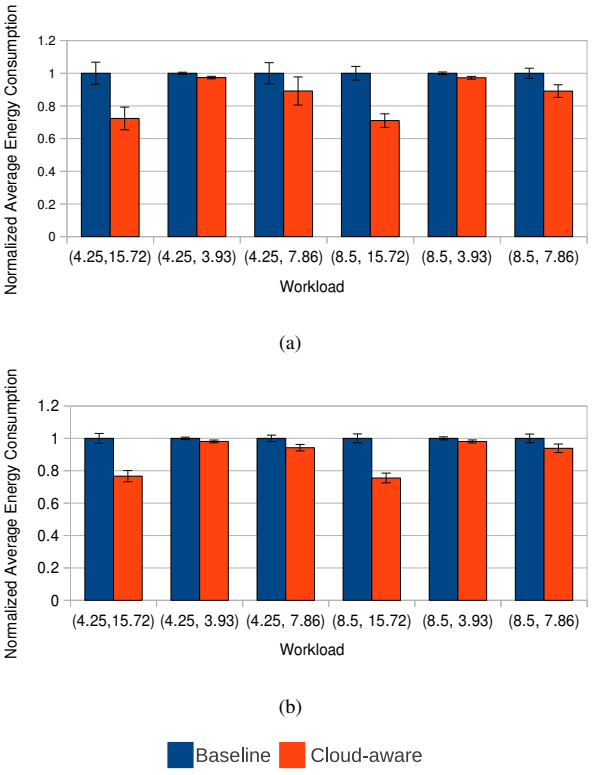

Fig. 2. Average energy consumption and standard deviation normalized by the energy consumption of the baseline algorithm. The workload is labeled as (α, β) parameters of the Weibull distribution that characterizes the inter-arrival time of jobs. (a) 20% of urgent requests (b) 50% of urgent requests.

TABLE II
AVERAGE REQUEST REJECTION RATES. THE WORKLOAD IS LABELED AS (α, β) PARAMETERS OF THE WEIBULL DISTRIBUTION THAT CHARACTERIZES THE INTER-ARRIVAL TIME OF JOBS.

Workload	Urgent requests=20%		Urgent requests=50%	
	Baseline	Cloud-aware	Baseline	Cloud-aware
(4.25,15.72)	29.6%	29.6%	27.5%	27.5%
(4.25, 3.93)	43.3%	43.2%	52.1%	51.8%
(4.25, 7.86)	29.6%	29.6%	33.0%	32.9%
(8.5, 15.72)	29.6%	29.6%	27.7%	27.7%
(8.5, 3.93)	41.7%	41.6%	50.4%	50.3%
(8.5, 7.86)	29.4%	29.4%	31.2%	30.9%

is not appropriate in cloud environments. This is because the actual energy consumption of a VM is linked to the state of other VMs sharing the host. If some mechanism is employed that allows VMs to learn energy consumption of other VMs in the same host, it would break the encapsulation enabled by cloud systems, and it could even be seeing as a form of security flaw, because understanding energy consumption of co-hosted VMs enables parties to estimate load/activities being carried out by other VMs, eventually without permission from the parties operating the co-hosted VMs.

VI. CONCLUSIONS AND FUTURE WORK

As the size of cloud data centers increases, the need for more energy-efficient use of such infrastructures becomes critical to enable sustainable utilization of such platforms. In this paper, we targeted the problem of energy-efficient execution of urgent, CPU-intensive Bag-of-Tasks applications in clouds. This type of application is found is domains such as disaster management and healthcare. We proposed a cloud-aware scheduling algorithm that applies DVFS to enable deadlines for execution of urgent CPU-intensive Bag-of-Tasks jobs to be met with reduced energy expenditure. Whilst existing approaches focus on other programming models, they do not address the issue of execution deadlines, or are not suitable for cloud platforms. Our approach is able to significantly reduce energy consumption of the cloud while not incurring any impact on the Quality of Service offered to users.

As future work, we will improve our algorithm to support other types of applications, such as workflows and MapReduce. This requires consideration on the energy consumption of network and storage equipments during the scheduling process. Another interesting research direction is the investigation of an energy-efficient algorithm in the context of hybrid cloud scenarios. Finally, we will also investigate the interplay between our approach and other energy-efficiency strategies such as server or workload consolidation.

REFERENCES

[1] R. Buyya, C. S. Yeo, S. Venugopal, J. Broberg, and I. Brandic, "Cloud computing and emerging IT platforms: Vision, hype, and reality for delivering computing as the 5th utility," *Future Generation Computer Systems*, vol. 25, no. 6, pp. 599–616, 2009.

[2] J. G. Koomey, "Growth in data center electricity use 2005 to 2010," Analytics Press, Oakland, USA, Report, 2011.

[3] K. V. Knyazkov, D. A. Nasonov, T. N. Tchurov, and A. V. Boukhanovsky, "Interactive workflow-based infrastructure for urgent computing," in *Proceedings of the 2013 International Conference on Computational Science (ICCS)*, 2013.

[4] R. Abbott and H. Garcia-Molina, "Scheduling real-time transactions," *ACM SIGMOD Record*, vol. 17, no. 1, pp. 71–81, 1988.

[5] S. Eyerman and L. Eeckhout, "Fine-grained DVFS using on-chip regulators," *ACM Transactions on Architecture and Code Optimization*, vol. 8, no. 1, pp. 1:1–1:24, 2011.

[6] J. L. March, J. Sahuquillo, S. Petit, H. Hassan, and J. Duato, "Power-aware scheduling with effective task migration for real-time multicore embedded systems," *Concurrency and Computation: Practice and Experience*, vol. 25, no. 14, pp. 1987–2001, 2013.

[7] S. Srikantaiah, A. Kansal, and F. Zhao, "Energy aware consolidation for cloud computing," in *Proceedings of the Workshop on Power aware computing and systems (HotPower)*, 2008.

[8] L. Tsai and W. Liao, "Cost-aware workload consolidation in green cloud datacenter," in *Proceedings of the 1st International Conference on Cloud Networking (CLOUDNET)*, 2012.

[9] G. A. Geronimo, J. Werner, C. B. Westphall, C. M. Westphall, and L. Defenti, "Provisioning and resource allocation for green clouds," in *Proceedings of the 12th International Conference on Networks (ICN)*, 2013.

[10] A. Beloglazov and R. Buyya, "Optimal online deterministic algorithms and adaptive heuristics for energy and performance efficient dynamic consolidation of virtual machines in cloud data centers," *Concurrency and Computation: Practice and Experience*, vol. 24, no. 13, pp. 1397–1420, 2013.

[11] G. von Laszewski, L. Wang, A. J. Younge, and X. He, "Power-aware scheduling of virtual machines in DVFS-enabled clusters," in *Proceedings of the IEEE International Conference on Cluster Computing (CLUSTER)*, 2009.

[12] Y. Li, Y. Liu, and D. Qian, "An energy-aware heuristic scheduling algorithm for heterogeneous clusters," in *Proceedings of the 15th International Conference on Parallel and Distributed Systems (ICPADS)*, 2009.

[13] G. L. T. Chetsa, L. Lefevre, J.-M. Pierson, P. Stolf, and G. D. Costa, "A runtime framework for energy efficient HPC systems without a priori knowledge of applications," in *Proceedings of the 18th IEEE International Conference on Parallel and Distributed Systems (ICPADS)*, 2012.

[14] X. Ruan, X. Qin, Z. Zong, K. Bellam, and M. Nijim, "An energy-efficient scheduling algorithm using dynamic voltage scaling for parallel applications on clusters," in *Proceedings of the 16th International Conference on Computer Communications and Networks (ICCCN)*, 2007.

[15] L. Wang, G. von Laszewski, J. Dayal, and F. Wang, "Towards energy aware scheduling for precedence constrained parallel tasks in a cluster with DVFS," in *Proceedings of the 10th International Conference on Cluster, Cloud and Grid Computing (CCGrid)*, 2010.

[16] Y. Tian, C. Lin, Z. Chen, J. Wan, and X. Peng, "Performance evaluation and dynamic optimization of speed scaling on web servers in cloud computing," *Tsinghua Science and Technology*, vol. 18, no. 3, pp. 298–307, 2013.

[17] K. H. Kim, W. Y. Lee, J. Kim, and R. Buyya, "SLA-based scheduling of bag-of-tasks applications on power-aware cluster systems," *IEICE Transactions on Information and Systems*, vol. E93-D, no. 12, pp. 3194–3201, 2010.

[18] L. M. Zhang, K. Li, D. C.-T. Lo, and Y. Zhang, "Energy-efficient task scheduling algorithms on heterogeneous computers with continuous and discrete speeds," *Sustainable Computing: Informatics and Systems*, vol. 3, no. 2, pp. 109–118, 2013.

[19] P. Barham, B. Dragovic, K. Fraser, S. Hand, T. Harris, A. Ho, R. Neugebauer, I. Pratt, and A. Warfield, "Xen and the art of virtualization," in *Proceedings of the 19th ACM Symposium on Operating Systems Principles (SOSP)*, 2003.

[20] W. Kim, M. S. Gupta, G.-Y. Wei, and D. Brooks, "System level analysis of fast, per-core DVFS using on-chip switching regulators," in *Proceedings of the 14th International Symposium on High Performance Computer Architecture (HPCA)*, 2008.

[21] A. Iosup, O. Sonmez, S. Anoep, and D. Epema, "The performance of bags-of-tasks in large-scale distributed systems," in *Proceedings of the 17th International Symposium on High Performance Distributed Computing (HPDC)*, 2008.

[22] R. N. Calheiros, R. Ranjan, A. Beloglazov, C. De Rose, and R. Buyya, "CloudSim: A toolkit for modeling and simulation of cloud computing environments and evaluation of resource provisioning algorithms," *Software: Practice and Experience*, vol. 41, no. 1, pp. 23–50, 2011.