Dynamic Management of CPU Resources Towards Energy Efficient and Profitable Datacentre Operation

Christos Kalogirou[0000−0003−0932−7782], Christos D. Antonopoulos[0000−0002−6486−062X], Spyros Lalis[0000−0003−2232−3559], and Nikolaos Bellas[0000−0002−9522−9136]

University of Thessaly, Volos, Greece
{hrkalogi,cda,lalis,nbellas}@uth.gr

Abstract. Energy reduction has become a necessity for modern datacentres, with CPU being a key contributor to the energy consumption of nodes. Increasing the utilization of CPU resources on active nodes is a key step towards energy efficiency. However, this is a challenging undertaking, as the workload can vary significantly among the nodes and over time, exposing operators to the risk of overcommitting the CPU. In this paper, we explore the trade-off between energy efficiency and node overloads, to drive virtual machine (VM) consolidation in a cost-aware manner. We introduce a model that uses runtime information to estimate the target utilization of the nodes to control their load, identifying and considering correlated behavior among collocated workloads. Moreover, we introduce a VM allocation and node management policy that exploits the model to increase the profit of datacentre operators considering the trade-off between energy reduction and potential SLA violation costs. We evaluate our work through simulations using node profiles derived from real machines and workloads from real datacentre traces. The results show that our policy adapts the nodes' target utilization in a highly effective way, converging to a target utilization that is statically optimal for the workload at hand. Moreover, we show that our policy closely matches, or even outperforms two state-of-the-art policies that combine VM consolidation with VFS – the second one, also operating the CPU at reduced voltage margins – even when these are configured to use a static, workload- and architecture-specific target utilization derived through offline characterization of the workload.

Keywords: Energy efficiency · Dynamic CPU management · Dynamic VM consolidation · Cost-effective datacentre operation.

1 Introduction

Modern datacentres have a significant energy footprint, which accounts for approximately 2% - 3% of the worldwide energy consumption [9]. In fact, CPUs are responsible for up to 60% of the total energy consumption of compute nodes [8].

Therefore, optimizing the energy efficiency of these nodes is a first-class concern for both designers and operators of large-scale datacentres.

Techniques such as virtual machine (VM) consolidation [5], [7] promote energy efficiency at the datacentre-level by creating opportunities to switch off nodes. The challenge is to balance between packing VMs to the nodes as densely as possible, and avoiding node overloads that will trigger Service Level Agreement (SLA) violation penalties. At the node-level, dynamic voltage and frequency scaling (VFS) [13], available on most modern processors, enables the dynamic, joint manipulation of CPU voltage (V) and frequency (f) to support potentially more power- and energy-efficient operating points, according to the characteristics of the workload. Our recent work in [15] goes a step further, exploiting CPU voltage guardbands to enable even more energy-efficient operation by configuring CPUs outside their normal (V, f) envelope, with only slightly increased crash probability (if managed carefully).

Dealing with this multi-parametric configuration and optimization space is far from trivial. Scheduling, configuration and resource management policies for the cloud often use static thresholds to drive their decisions [15], [4] with node target utilization thresholds being a typical example. This popular approach has low design complexity, however (a) selection of the optimal static threshold requires *a priori* knowledge of the workload, which may not always be realistic, (b) it fails to capture inter-node variations of workload characteristics at any given time, (c) it fails to capture system-wide workload variations in time, and (d) the value of the optimal threshold is also sensitive to hardware characteristics and configuration.

In this paper, we tackle the aforementioned weaknesses with a policy which dynamically adapts target node utilization thresholds. This is achieved by analyzing the workload characteristics observed in the past and predicting short-term future behavior. Moreover, we quantify the correlation among collocated workloads on each node in order to identify a sweet-spot between high node utilization and risk of SLA violation penalties due to unexpected load increases, which may lead to node overloads. More specifically:

- We introduce a scalable, analytic and architecture-agnostic model to estimate target CPU utilization for individual nodes in a datacentre, exploiting knowledge on the past behaviour of the VMs scheduled on these nodes. These adaptive utilization targets favor high VM consolidation, while controlling the risk of node overloads.
- We find that, in order to achieve this dual goal, the policy needs to identify correlations between the CPU capacity requests of co-scheduled VMs and consider them when deciding on node utilization targets.
- We introduce an adaptive VM management and node configuration policy that exploits the aforementioned adaptive thresholds for VM consolidation, together with VFS and with CPU configuration at reduced voltage margins, considering the cost of SLA violation penalties due to both node overloads and crashes (due to operation with reduced margins).

– We perform simulations to evaluate our approach using real datacentre workload traces and realistic node parameters ((V, f) steps , voltage margins, power consumption to the plug, failure probability) derived from real systems with two different processor architectures, an Intel Xeon E3-1120 v5, and an ARM-based Ampere Computing X-Gene 3.

To the best of our knowledge, this is the first work that combines per node target CPU utilization adaptivity with VFS and relaxed CPU voltage guardbands in a cost-centric approach, to increase the profit of cloud infrastructure providers exploring the trade-off between energy cost and potential SLA violation penalties. Our results show that our adaptive policy manages to always outperform two state-of-the-art policies using optimal target CPU utilization values derived via pre-characterization of the workload at hand on the respective architecture: (a) a policy that combines VM consolidation with VFS [3], and (b) a policy which also exploits reduced CPU voltage guardbands on top of VM consolidation and VFS [15].

2 Background

In this section, we briefly outline the notation and some basic assumptions used in our adaptive model introduced in Section 3. We use a notation and assumptions compatible to those in [15].

2.1 System model

We assume a datacentre with nodes $n_i, 1 \leq i \leq N$. Each node has limited memory MEM_i. All nodes have the same CPU, which can operate at different nominal voltage-frequency points (V, f), where (V_{max}, f_{max}) offers maximum performance. For each nominal point, there is a reduced voltage configuration (V^r, f) with $V^r < V$, offering the same performance at lower power consumption. However, in this case the node may crash with probability $Pfail > 0$, whereas $Pfail = 0$ for nominal configurations.

Different jobs are submitted for execution, packaged as virtual machines $VM_m, 1 \leq m \leq M$ with formally declared resource requirements in respective SLAs. Let MEM_m^{SLA} be the required memory and $C_{m,f_{max}}^{SLA}$ be the CPU capacity at f_{max} for VM_m as per its SLA. Notably, when the CPU operates at $f < f_{max}$, the effective CPU capacity as per the SLA becomes $C_{m,f}^{SLA} \geq C_{m,f_{max}}^{SLA}$ (the increase depends on the sensitivity of the VM to frequency scaling).

We partition time into periods and assume that VMs arrive and terminate at their boundaries. Further, we divide each period in K timeslots of duration $slotT$ (the total duration of a period is $prdT = K \times slotT$). Let $C_{m,f,p}^{req}$ be the actual CPU capacity requirements of VM_m for frequency f throughout period p. Note that $C_{m,f,p}^{req} \leq C_{m,f}^{SLA}$, i.e., in some periods the VM may not need the full CPU capacity as per its SLA.

Let $C_{m,f,p,k}^{alloc}$ be the CPU capacity at frequency f that is allocated to VM_m in the k^{th} timeslot of p. If the host node is overloaded, the VM may get less capacity

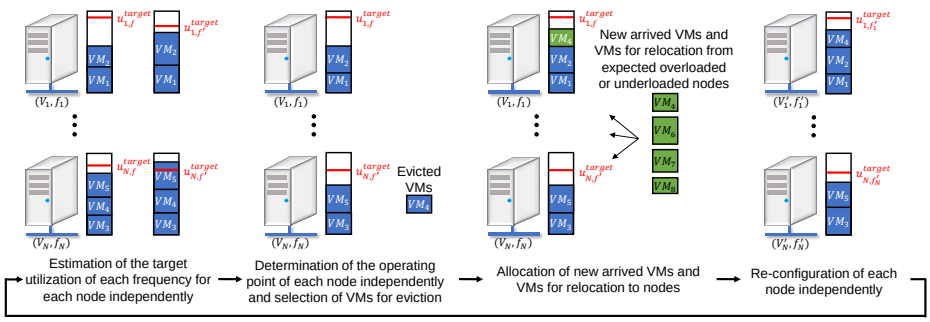

Fig. 1: Overview of the proposed approach.

than needed, $C^{alloc}_{m,f,p,k} < C^{req}_{m,f,p}$. Furthermore, the restart of VM_m on a node in period p (due to an eviction from its old host) takes R_m time slots (linearly dependent contractually agreed memory footprint of VM_m) thus $C^{alloc}_{m,f,p,k} = 0, 1 \leq k \leq R_m$. We assume VMs to be stateless, therefore they can be restarted without needing elaborate migration. Also, if the host of VM_m crashes (due to operation with reduced voltage), $C^{alloc}_{m,f,p,k} = 0, 1 \leq k \leq K$ for all slots of the period.

2.2 VM allocation approach

When a period ends, the target utilization, VM allocation and operating point is decided for each node, so as to minimize the expected cost for the datacentre due to the node's energy consumption and SLA violation penalties – the latter apply in case of node overload as well as in case the VM remains unavailable because it is being restarted on another node or the host node crashes.

Our approach is divided in the following main steps, illustrated in Figure 1. (1) We estimate the target node utilization for each CPU frequency based on the current VM allocation and load, in order to identify expected node overloads for each operating point and pick the VMs that need to be evicted from the node to avoid overload. (2) We select the best nominal operating point of each node considering the trade-off between the node's energy cost and SLA violation penalties due to restart of the evicted VMs on other nodes. (3) We allocate evicted VMs and newly arrived VMs to nodes, respecting the selected operating point and target utilization of each node. At this point, we consider activating additional nodes or deactivating underutilized nodes. (4) We re-configure each node considering the updated VM allocation, exploiting operation at reduced voltage margins for additional cost gains selecting the operating point in a cost-driven manner. Note that steps (1), (2) and (4) are performed using only local data of the node. Thus, the approach is scalable even for large datacentres.

3 Node-level estimations

We introduce a model that considers the trade-off between the energy cost and potential SLA violation penalties. In contrast with previous work [15], this model does not require in advance knowledge and characterization of the workload, but instead quantifies and adapts to workload characteristics at execution time. Moreover, it estimates the cost at the level of the entire system, even though scheduling decisions are taken at the node level. Hence, our approach is: a) easily applicable in realistic settings, b) scalable, and c) cost-effective even for large datacentres. In this section we discuss in detail the estimation of the different costs involved, whereas in Section 4 we focus on the adaptation to workload characteristics.

3.1 Target CPU utilization and VM evictions

Each node n_i has a target CPU utilization $u_{i,f,p}^{target}$ for n_i in p for CPU frequency f. When the period ends, the target utilization is adjusted for the next period $p+1$ based on the current VM allocation and a tentative frequency f_i, possibly selecting some VMs for eviction to avoid overloads.

The tentative VM evictions are captured via matrix $Evct$, where $Evct[i, f_i, m, p+1] = 1$ if VM_m is to be evicted from n_i assuming operation at f_i in $p+1$, else 0. Similarly, we use $Rem[i, f_i, m, p+1]$ to encode whether VM_m will remain on n_i in $p+1$. This exploration is done for each of the supported CPU frequencies, in order to find the best option for n_i. Note that any VM allocation must respect the full memory requirements of each hosted VM, i.e., $\sum_{m=1}^{M} Rem[i, *, m, p+1] \times MEM_m^{SLA} \leq MEM_i$.

3.2 Load estimation

We estimate the CPU requirements of each VM_m for frequency f in the next period $p+1$ as $\hat{C}_{m,f,p+1}^{req} = \overline{C}_{m,f,p}^{req}$, where $\overline{C}_{m,f,p}^{req}$ are the VM's mean CPU requirements over the past periods up to p adjusted to f.

Then, the estimated load for n_i assuming operation at f_i, based on the VMs that will remain on the node in the next period, is calculated as

$$\hat{load}_{i,f_i,p+1}^{rem} = \sum_{m=1}^{M} Rem[i, f_i, m, p+1] \times \hat{C}_{m,f_i,p+1}^{req} \quad (1)$$

We also estimate the load of each VM to be evicted from n_i and restarted on another node. However, we do not know the hosts of these VMs nor the respective CPU frequencies in the next period. As an approximation, we estimate the load of the evicted VMs for each supported frequency f as

$$\hat{load}_{i,f_i,p+1,f}^{vms} = \sum_{m=1}^{M} Evct[i, f_i, m, p+1] \times \hat{C}_{m,f,p+1}^{req} \quad (2)$$

And then we estimate the total load of the VMs to be evicted from n_i for the entire system, by weighing the above load for each frequency according to the nodes' operating frequencies during the previous period p

$$\hat{load}^{evct}_{i,f_i,p+1} = \sum_{f \in F} \frac{nodes_{f,p}}{nodes_p} \times \hat{load}^{vms}_{i,f_i,p+1,f} \tag{3}$$

where $nodes_{f,p}$ is the number of nodes that operated at f in p, and $nodes_p$ is the number of nodes that were active in p.

Thus, the total estimated VM load related to n_i (for the node and the system as a whole) assuming it will operate at f_i in the next period $p+1$, is

$$\hat{load}_{i,f_i,p+1} = \hat{load}^{rem}_{i,f_i,p+1} + \hat{load}^{evct}_{i,f_i,p+1} \tag{4}$$

3.3 Energy cost estimation

The estimated energy cost of each active node n_i that will host one or more VMs in the next period, assuming it will be configured to operate at (V_i,f_i), is calculated as

$$\hat{costE}^{rem}_{i,V_i,f_i,p+1} = P_{node}(V_i, f_i, \hat{u}_{i,f_i,p+1}) \times prdT \times price_{en} \tag{5}$$

$$P_{node}(V, f, u) = A + B \times f \times V^2 \times u \tag{6}$$

$$\hat{u}_{i,f_i,p+1} = min(\hat{load}^{rem}_{i,f_i,p+1}, 1) \tag{7}$$

where $price_{en}$ is the energy price and $\hat{u}_{i,f_i,p+1}$ is the estimated node utilization for the next period based on the estimated load as discussed in Section 3.2. Function $P_{node}(V, f, u)$ returns the node's power consumption when the CPU operates at (V,f) with utilization u, based on a linear power estimation model [15] where A and B are platform-specific constants for the node's static and dynamic power consumption, respectively.

The energy cost due to the load of the VMs that will be evicted from n_i is approximated as follows. First, the individual cost of each evicted VM_m if hosted on a node configured to operate at (V,f) is

$$\hat{costE}^{vm}_{i,m,f_iV,f,p+1} = P_{vm}(V, f, \hat{C}^{req}_{m,f,p+1}) \times prdT \times price_{en} \tag{8}$$

$$P_{vm}(V, f, u) = \frac{u}{\overline{u}^{target}_{f_i,p}} \times A + B \times f \times V^2 \times u \tag{9}$$

In this case, the CPU utilization factor for the dynamic component of the power estimation function is set to the expected CPU load of VM_m in the next period $\hat{C}^{req}_{m,f,p+1}$. We also factor the static power consumption component with the ratio of that load to the mean target CPU utilization of all nodes configured to operate at f_i in the previous period p, let $\overline{u}^{target}_{f_i,p}$, as a proxy for the likelihood of that particular VM eviction leading to the activation of a new node in $p+1$.

Then, we estimate the total energy cost for the entire system of all VMs that will be evicted from n_i, as

$$\hat{cost}E^{evct}_{i,V_i,f_i,p+1} = \sum_{(V,f)\in VF} (\frac{nodes_{V,f,p}}{nodes_p} \times \sum_{m=1}^{M} Evct[i,f_i,m,p+1] \times \hat{cost}E^{vm}_{i,m,f_i,V,f,p+1}) \quad (10)$$

weighing the contribution of each operating point in the same spirit this done for the load estimation in Equation 3.

Thus, the total estimated energy cost related to n_i (for the node and the system as a whole) assuming it will operate at (V_i,f_i) in the next period $p+1$ is

$$\hat{cost}E_{i,V_i,f_i,p+1} = \hat{cost}E^{rem}_{i,V_i,f_i,p+1} + \hat{cost}E^{evct}_{i,V_i,f_i,p+1} \quad (11)$$

3.4 SLA violation cost estimation

We estimate the CPU capacity $\hat{C}^{alloc}_{i,m,f_i,p+1,k}$ allocated to VM_m hosted on n_i in the k^{th} timeslot of period $p+1$ as follows. If n_i $\hat{load}^{rem}_{i,f_i,p+1} \leq 1$ then $\hat{C}^{alloc}_{i,m,f_i,p+1,k} = \hat{C}^{req}_{i,m,f_i,p+1}$, else (if the node is overloaded) $\hat{C}^{alloc}_{i,m,f_i,p+1,k} < \hat{C}^{req}_{i,m,f_i,p+1}$ since the node's CPU capacity is divided among the VMs proportionally to their requirements (all VMs have the same priority). Also, $\hat{C}^{alloc}_{i,m,f_i,p+1,k} = 0$ in timeslots where VM_m is restarting as well as in all timeslots of the period if n_i crashes.

Whenever VM_m does not get the requested CPU capacity, the datacentre provider has to pay a penalty to the customer. We consider the *Percentage Price Refund* (PPR) model [11] where the penalty is a percentage of the VM price for the duration of the violation, depending on the percentage of the CPU capacity that was not allocated w.r.t. to the SLA requirements. More specifically, the expected violation cost for VM_m for the k^{th} timeslot of period $p+1$ is

$$\hat{cost}V^{vm}_{m,f_i,p+1,k} = \begin{cases} q_m \times \frac{C^{SLA}_{m,f_i} - \hat{C}^{alloc}_{m,f_i,p+1,k}}{C^{SLA}_{m,f_i}} \times price_m, \\ \text{if } \hat{C}^{alloc}_{m,f_i,p+1,k} < \hat{C}^{req}_{m,f_i,p+1} \\ 0, \text{if } \hat{C}^{alloc}_{m,f_i,p+1,k} = \hat{C}^{req}_{m,f_i,p+1} \end{cases} \quad (12)$$

where q_m reflects the severity of the violation and $price_m$ is the price charged for executing VM_m per timeslot.

We then calculate the total estimated SLA violation cost for n_i for the entire period $p+1$, as follows

$$\hat{cost}V_{i,f_i,p+1} = \sum_{m=1}^{M}\sum_{k=1}^{K} Rem[i,f_i,m,p+1] \times \hat{cost}V^{vm}_{m,f_i,p+1,k} \quad (13)$$

This equation captures node overloads and VM restarts during normal operation, let $\hat{cost}V^{nofail}_{i,f_i,p+1}$, as well as the case where n_i crashes and does not run any VM, let $\hat{cost}V^{fail}_{i,f_i,p+1}$.

3.5 Total estimated node cost

Based on the above, the total estimated operational cost related to n_i (for the node and the system as a whole) assuming it will operate at (V_i,f_i) in the next period $p+1$ is

$$\hat{cost}_{i,V_i,f_i,p+1} = Pfail \times \hat{costV}^{fail}_{i,f_i,p+1} + \\ (1 - Pfail) \times (\hat{costE}_{i,V_i,f_i,p+1} + \hat{costV}^{nofail}_{i,f_i,p+1}) \tag{14}$$

calculated by weighing the cost of normal node operation vs a node crash with the corresponding probabilities. We assume that if a node fails it does so for an entire period during which it does not run any VMs and does not consume any energy.

4 Adaptive target CPU utilization

The target node utilization for the next period $u^{target}_{i,f,p+1}$ is set with some slack, so that the node can handle load beyond the estimated $\hat{C}^{req}_{i,f_i,p+1}$ without being overloaded. This is subject to an interesting trade-off: High utilization targets result to more energy-efficient hence also greener and lower cost operation. However, if future CPU requests of the VMs are underestimated and overshoot the CPU capacity of the node, SLA violations will occur, incurring the respective penalties for the operator.

In addition, potential correlation between different collocated VMs further perplexes the estimation of a good CPU utilization target. Let $C^{ratio}_{m,p} = \frac{C^{req}_{m,f_{max},p}}{C^{SLA}_{m,f_{max}}}$ be the percentage of the CPU requirements of VM_m in period p with respect to the contractually promised CPU capacity, both at frequency f_{max}. Figure 2 illustrates the C^{ratio} of eight VMs from Google traces [19] over 10 periods. Each line corresponds to a different VM. One can identify clusters of VMs with similar, positively correlated $C^{ratio}_{m,p}$ variations (lines of the same color). Moreover, VMs in the blue and green cluster are characterized by a rapid increase of their requested CPU capacity at period 10, while the red cluster has the opposite trend to the blue one.

If decisions on target CPU utilization do not consider such phenomena, they are highly likely to lead to node overloads. On the one hand, the identification of positively correlated VMs should encourage more conservative values of target CPU utilization. On the other hand, the co-execution of anti-correlated VM clusters on the same node can protect against overloads due to dynamic workload changes: when one cluster increases its requests, its counterpart tends to decrease them, thus balancing the total load on the host node. This allows for more aggressive (higher) CPU utilization targets, and a more energy-efficient usage of the available resources.

To address the aforementioned challenges, at period boundaries we evaluate, separately for each node n_i, the past behavior of its assigned VMs and decide a

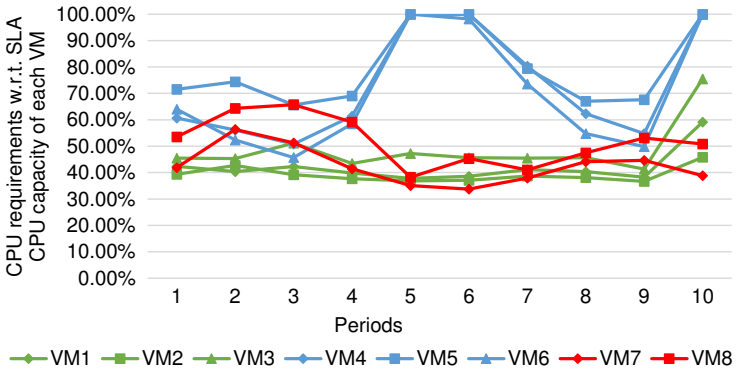

Fig. 2: CPU pressure of different VMs w.r.t their contractual (SLA) maximum CPU requirements, over consecutive periods.

new value for $u^{target}_{i,f,p+1}$ for the following period. The estimation process involves the following steps: (1) For every pair (VM_l, VM_m) of VMs on the node we quantify the Pearson correlation $corr_{(l,m),p}$ between the C^{ratio} of VM_l and VM_m during periods up to p. (2) We use linear regression to estimate the CPU capacity requests of every VM_m on the node, given the capacity requests of a different VM_l for the next period, according to $corr_{(l,m),p}$. (3) We pessimistically assume that on the next period a specific VM_l will request its maximum contractually agreed CPU capacity, i.e., $\hat{C}^{ratio}_{l,p+1} = 1$. Given this assumption and the correlations determined in step (2), we reach a conservative estimation of the respective total node load. (4) We repeat steps (2)-(3) using each VM on the node as the baseline for the estimation, and then use the highest (most pessimistic) load estimation for period $p+1$ to set the new target CPU utilization for the node. Below we provide more details about these steps.

In step (2) we first apply linear regression, as modelled by Equation 15, to predict the C^{ratio} of VM_m in period $p+1$ assuming we know the C^{ratio} of VM_l on the same period:

$$\hat{C}^{ratio}_{(m|l),p+1} = \alpha_{(m|l),p} \times \hat{C}^{ratio}_{l,p+1} + \beta_{(m|l),p} + \epsilon_{(m|l),p} \tag{15}$$

where $\alpha_{(m|l),p}$ is the slope of the regression line, $\beta_{(m|l),p}$ is the intercept, and $\epsilon_{(m|l),p}$ is the error of the prediction. We calculate $\alpha_{(m|l),p} = corr_{(m,l),p} \times \frac{std_{m,p}}{std_{l,p}}$, where $std_{l,p}$ and $std_{m,p}$ are the standard deviations of the C^{ratio} of VM_l and VM_m respectively, over past periods up to p. To derive $\beta_{(m|l),p}$, we first calculate the mean C^{ratio} of VM_m and VM_l up to period p, let $\overline{C}^{ratio}_{m,p}$ and $\overline{C}^{ratio}_{l,p}$, respectively. Then, we set $\epsilon_{(m|l),p} = 0$ and solve Equation 15 for $\beta_{(m|l),p}$, using $\overline{C}^{ratio}_{m,p}$ and $\overline{C}^{ratio}_{l,p}$ in place of $\hat{C}^{ratio}_{(m|l),p+1}$ and $\hat{C}^{ratio}_{l,p+1}$. Finally, since our goal is to avoid node overloads, we conservatively set $\epsilon_{(m|l),p}$ to the maximum absolute error between the predicted values of our model and the observed real values over past periods.

In step 3, we predict the load of n_i for period $p+1$, as a fraction of its total CPU capacity, assuming $\hat{C}^{ratio}_{l,p+1} = 1$, as

$$\hat{load}^{pre}_{i,l,f_i,p+1} = C^{SLA}_{l,f_i} + \sum_{m \neq l} Rem[i, f_i, m, p+1] \times \hat{C}^{pre}_{(m|l),f_i,p+1} \tag{16}$$

where $\hat{C}^{pre}_{(m|l),f_i,p+1}$ is the predicted CPU capacity request by VM_m for frequency f_i under the assumption that $\hat{C}^{ratio}_{l,p+1} = 1$. This is calculated by deriving the predicted CPU capacity request $\hat{C}^{pre}_{(m|l),f_{max},p+1} = \hat{C}^{ratio}_{(m|l),f_{max},p+1} \times C^{SLA}_{m,f_{max}}$ at f_{max} and then calculating $\hat{C}^{pre}_{(m|l),f_i,p+1}$ for f_i (according to the sensitivity of VM_m to frequency scaling).

Finally, in step 4, after performing the above for each VM hosted on n_i, we determine the highest (most pessimistic) load across all scenarios, let $\hat{load}^{max}_{i,f_i,p+1}$. We then use this value to set the new target CPU utilization for n_i at f_i in $p+1$:

$$u^{target}_{i,f_i,p+1} = u^{target}_{i,f_i,p} \times (1 + \frac{1 - \hat{load}^{max}_{i,f_i,p+1}}{\hat{load}^{max}_{i,f_i,p+1}}) = \frac{u^{target}_{i,f_i,p}}{\hat{load}^{max}_{i,f_i,p+1}} \tag{17}$$

where we adjust the previous target based on the relative slack $\frac{1-\hat{load}^{max}_{i,f_i,p+1}}{\hat{load}^{max}_{i,f_i,p+1}}$ between the most pessimistic load and full node utilization, asymptotically pushing the maximum expected CPU requests on the node close to (yet lower than) full utilization. A negative slack indicates that the node is at risk for an overload, thus the target utilization is lowered. If, on the other hand, the slack is positive, there is CPU capacity available for hosting more VMs, and the target utilization is increased. As an exception, if $u^{target}_{i,f_i,p+1} > \hat{load}^{max}_{i,f_i,p+1}$, we set $u^{target}_{i,f_i,p+1} = u^{target}_{i,f_i,p}$. This guards against the corner-case of having enough spare CPU capacity on n_i just because it did not host "enough" VMs in p, as in this case a further increase of the target utilization would introduce the risk of excessive arrival of additional VMs on n_i, leading to an overload.

5 VM Allocation and Node Configuration Policy

In this section, we introduce Adaptive Target Utilization and Configuration ($ATUC$), a VM allocation policy which dynamically adjusts the target utilization and configuration of each node to increase the profit of datacentre providers. The optimization target is to minimize the predicted cost-to-load ratio of each node for the next period

$$\hat{CL}^{ratio}_{i,V_i,f_i,p+1} = \frac{\hat{cost}_{i,V_i,f_i,p+1}}{\hat{load}_{i,f_i,p+1}} \tag{18}$$

where the nominator and denominator of the fraction comes from Equation 14 and Equation 4, respectively. Note that this involves only local information, as discussed in Section 3.2, Section 3.3 and Section 3.4.

The *ATUC* policy consists of the following main steps:

1) Estimation of target CPU utilization for each frequency: We use the methodology discussed in Section 4 to estimate the new target utilization $u^{target}_{i,f_i,p+1}$ for the next period $p+1$, for each node n_i and supported CPU frequency f_i.

2) Cost-driven node configuration and VM eviction: Given $u^{target}_{i,f_i,p+1}$ for each f_i, we select the best operating point for n_i in $p+1$ and the VMs to be evicted in this case. At this point, we focus on VM (re)assignment and only consider nominal operating points ($Pfail = 0$). The decision whether to configure the node to operate at reduced voltage margins (where $Pfail > 0$) is taken in the last step of the policy.

Algorithm 1 Selection of nominal operating point (V'_i,f'_i) and evicted VMs for n_i in the next period $p+1$.

Input: $vmsHost_{i,p}$, $u^{target}_{i,*,p+1}$
Output: (V'_i, f'_i), $vmsEvct_{i,p+1}$

$minCLRatio \leftarrow \infty$
for each nominal (V_i, f_i) and $u^{target}_{i,f_i,p+1}$ **do**
  $Rem[i, f_i, m, p+1] \leftarrow 1, \forall m \in vmsHost_{i,p}$
  $Evct[i, f_i, m, p+1] \leftarrow 0, \forall m \in vmsHost_{i,p}$
  $\hat{load}^{rem}_{i,f_i,p+1} \leftarrow$ calculate based on Equation 1
  while $\hat{load}^{rem}_{i,f_i,p+1} > u^{target}_{i,f_i,p+1}$ **do**
    $minRestartCost \leftarrow \infty$
    for each $m : Rem[i, f_i, m, p+1] = 1$ **do**
      $vmRestartCost \leftarrow \sum_{k=1}^{R_m} \hat{cost}V^{vm}_{m,f_i,p+1,k}$
      if $vmRestarCost < minRestartCost$ **then**
        $minRestartCost \leftarrow vmRestartCost$
        $vmEvct \leftarrow m$
      end if
    end for
    $Rem[i, f_i, vmEvct, p+1] \leftarrow 0$
    $Evct[i, f_i, vmEvct, p+1] \leftarrow 1$
    $\hat{load}^{rem}_{i,f_i,p+1} \leftarrow$ calculate based on Equation 1
  end while
  $\hat{CL}^{ratio}_{i,V_i,f_i,p+1} \leftarrow$ calculate based on Equation 18
  if $\hat{CL}^{ratio}_{i,V_i,f_i,p+1} < minCLRatio$ **then**
    $minCLRatio \leftarrow \hat{CL}^{ratio}_{i,V_i,f_i,p+1}$
    $(V'_i, f'_i) \leftarrow (V_i, f_i)$
  end if
end for
$vmsEvct_{i,p+1} \leftarrow \{m : Evct[i, f'_i, m, p+1] = 1\}$

The high-level logic of this step is given in Algorithm 1 (while some values can be calculated incrementally, we refer to the previously introduced equations

for more clarity). The algorithm takes as input the VMs hosted on n_i in the previous period p, and the node utilization targets for each CPU frequency in the next period $p+1$, and produces as output the nominal operating point for the next period $p+1$ and the VMs to be evicted from n_i. The logic is briefly as follows. For each nominal CPU operating point, starting from the previous VM allocation (lines 3-4) and the corresponding estimated load (line 5), the VMs with the smallest SLA violation cost due to restart on another node (lines 7-14) are tentatively selected for eviction (lines 15-16) and the expected node load is recalculated (line 17), until the load does not exceed the target utilization. Note that the evicted VMs are assumed to be hosted on a non-overloaded node (the cost only comes from the time slots needed to restart the VM). Then, the operating point in question is preferred if the respective cost-to-load ratio is better than the one of the current selection (19-22). The final (best) selection results after performing this check for all nominal operating points.

3) Assignment of unallocated VMs to nodes and switch-off of underutilized nodes At this point there are two types of unallocated VMs that need to be assigned to nodes: the VMs selected for eviction in the previous step and newly arrived VMs (if any). In a first step, we use the approach in [15] (best fit decreasing allocation) to assign unallocated VMs to nodes. In a second step, we adopt the technique in [5] to identify severely underutilized nodes and switch them off, by relocating the VMs accommodated on them (as above).

4) Cost-driven re-configuration of nodes at a nominal or reduced voltage margins operating points: Finally, based on the new VM allocation, the CPU of each node is configured to the most cost-effective operating point. This is done by examining all (V, f) configurations that do not violate the target utilization, and selecting the one that minimizes the respective cost-to-load ratio according to Equation 18. In this step, reduced voltage configurations are considered as well, taking into account the probability of node crash and the respective SLA violation costs as per Equation 14.

6 Experimental Evaluation

In this section, we evaluate the *ATUC* policy via simulations. For these experiments, we use real-world traces and hardware parameters derived from real platforms. We also compare *ATUC* against [3] (*LrMmt*), which combines VM consolidation with VFS, and [15] (*XM-VFS*), which exploits configuration of reduced CPU voltage margins on top of these two techniques. Both these policies use a static target CPU utilization value, which is derived by offline characterization of the workload and the platform architecture at hand.

6.1 Experimental Setup

For our simulations, we use the CloudSim toolkit [6], a popular framework for evaluating cloud policies, which already supports mechanisms such as VM consolidation and configurable node hardware parameters. We extend CloudSim to

implement our system model (Section 3) and the *ATUC* policy (Section 5), including target utilization adaptivity at the node granularity (Section 4). We use real values for the energy price [2] ($0.14 per KWh) and the range price of the VMs [1] ($\$0.0062 - \$0.896$ per hour). Our simulated workload consists of $10,000$ VMs from Google traces [19], for a period of one day. The workload has a mean cumulative CPU load of 0.86 and a standard deviation of 0.12 (the values are normalized to the maximum cumulative CPU load during the simulated period). Finally, we assume the policy is invoked every 300 seconds.

6.2 Hardware parameters

We configure the simulator with the hardware parameters of two real systems with different processor architectures, namely an Intel Xeon E3-1220 v5 and an ARM-based Ampere Computing X-Gene 3. In order to enable a direct and fair comparison between *ATUC* and *XM-VFS*, we use the parameters identified by the characterization of these two systems in [15], in terms of (a) the reduced voltage margins of the processors, (b) the failure probability when operating at reduced margins configurations, (c) a power model that estimates the power at the plug for different utilization levels and operating points, and (d) the performance sensitivity of applications due to frequency scaling. More specifically, Table 1 gives the parameters of the (V, f) pairs (nominal and reduced margins) used for the Intel and the ARM platforms. The failure probability for a scheduling period of 300 seconds, when the processors operate at reduced voltage margins configurations, is $Pfail = 0.000579$. Similarly power at the plug, is calculated using linear model $A + B * V^2 * f * u$, where A and B are platform-specific constants of the model, V is the voltage, f is the frequency and u is the utilization of the processor. For the Intel platform $A = 34.01$ and $B = 18.98$, whereas for the ARM platform $A = 52.48$ and $B = 34.38$ [15].

Table 1: Intel Xeon E3-1220 v5 and ARM-based Ampere Computing X-Gene 3 operating points.

Intel platform			ARM platform		
Frequency (GHz)	Nominal Voltage (mV)	Reduced Voltage (mV)	Frequency (GHz)	Nominal Voltage (mV)	Reduced Voltage (mV)
2.0	850	666	0.4	880	790
2.5	922	741	1.3	880	790
3.0	1075	865	2.2	880	830
3.3	1147	929	3.0	880	840

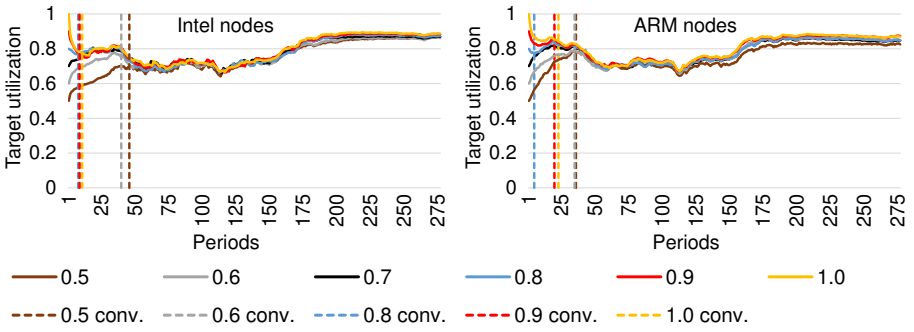

Fig. 3: $\overline{u^{target}}$ per period for different initial values of u^{target} $(0.5 - 1.0)$.

6.3 Threshold and cost convergence

ATUC adjusts the target utilization of each node independently, adapts VM allocation to nodes, and hardware configuration according to the dynamic workload requirements, in order to minimize the cost for the operator. It is important for *ATUC* to be resilient to sub-optimal initial values of the target utilization (u^{target}), as historical data of VM behavior are not available during the first periods of the simulations.

Figure 3 presents $\overline{u^{target}}$ (the mean target utilization of the nodes) for each period for different initial u^{target} values (different solid lines). The dashed lines mark (for the initial u^{target} with the same color) the time of convergence with the $\overline{u^{target}}$ timeseries for an initial target utilization of 0.7. We select 0.7 as the baseline for the convergence study as it is roughly in the middle of the range of evaluated initial target utilization values.

We observe that *ATUC* is resilient to the "sub-optimal" choice of the initial value of u^{target}, as it eventually converges to similar $\overline{u^{target}}$ values in all cases. However, the convergence time depends on that initial value. Lower initial values (0.5 and 0.6) converge later, as *ATUC* is conservative when increasing the target utilization, in order to avoid node overloads. For example, recovering from a low initial u^{target} of 0.5 requires 46 and 36 periods for the Intel and ARM platform, respectively. On the other hand, the convergence from higher initial values of u^{target} (0.8, 0.9 and 1.0) is quick, as *ATUC* decreases the target utilization rapidly when at risk for node overloads. As an example, $\overline{u^{target}}$ estimations reached with initial $u^{target} = 1.0$ and 0.7 (baseline) converge within just 11 and 23 periods for the Intel and ARM nodes, respectively. Overall, *ATUC* adjusts the target utilization within the ranges of 0.64 - 0.89 and 0.64 - 0.88 for the Intel and the ARM, respectively.

Apart from the resilience of $\overline{u^{target}}$ estimates to the initial value, we also study the cost breakdown per period for different initial u^{target} values to evaluate the cost of convergence. We perform experiments for low (0.5), high (1.0) and the baseline (0.7) initial values for u^{target}. Figure 4 illustrates the results, where the green, yellow, blue and red areas represent the cost of energy, and the cost of SLA

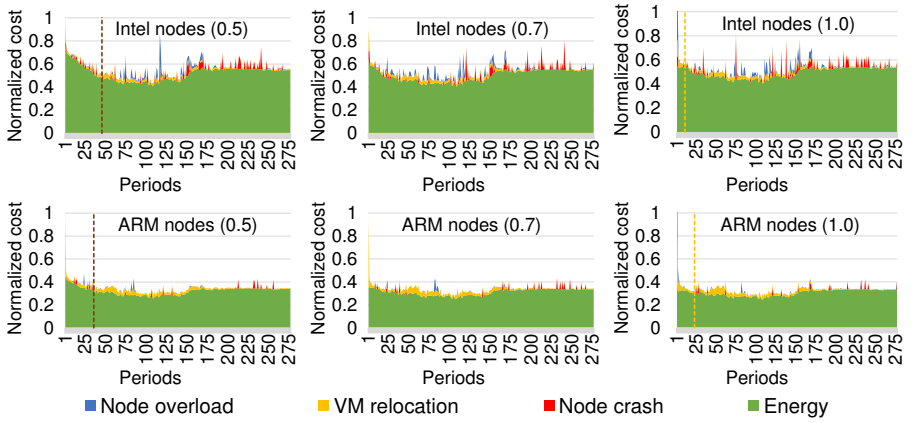

Fig. 4: Normalized cost per period for initial values of target utilization of 0.5, 0.7 and 1.0.

violations due to VM relocation, node overloads and node crashes, respectively. All costs have been normalized, for each of the two architectures (Intel and ARM), w.r.t. the total cost during the first period of execution with an initial $u^{target} = 0.5$. Similarly to the previous figure, for the experiments with initial $u^{target} = 0.5$ and 1.0 we use vertical dashed lines to illustrate the convergence time of the target utilization timeseries w.r.t to that for 0.7.

As expected, the case of an initial u^{target} of 0.5 introduces a higher cost (dominated by the energy cost) for both platforms, compared with the other two cases (0.7 and 1.0) for the periods before convergence of the target utilization, as the resulting low adaptive target utilization leads to the activation of more nodes. On the other hand, although a high u^{target} (1.0) achieves the lowest energy cost at the first periods, the total cost is higher due to node overloading. However, once convergence is reached, we observe that the energy cost pattern is similar, irrespective of the initial u^{target} selection. More specifically, after convergence, the relative energy cost difference between algorithm invocations with initial estimates 0.5 and 1.0, and the invocation with 0.7 is in the order of 1.76% and 1.55% respectively for the Intel platform. The respective relative energy cost differences for the ARM platform are 2.3% and 1.65%. In conclusion, *ATUC* manages to recover from sub-optimal initial values of the target utilization, asymptotically introducing similar energy cost per period, irrespective of the value of target utilization used for the initialization of the adaptive estimation algorithm. In the next section, we focus on the total cost, also including SLA violation penalties.

6.4 *ATUC* total cost of operation

As we discussed in Section 6.3, *ATUC* manages to recover from sub-optimal initial values of u^{target} achieving, after convergence, a low relative energy cost difference w.r.t. the baseline initial u^{target} case. In this section, we evaluate the

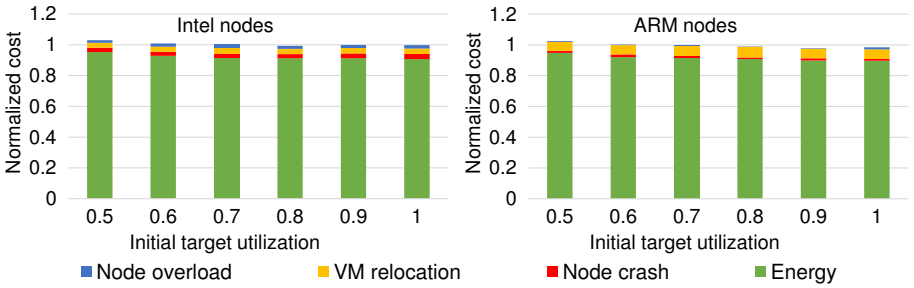

Fig. 5: Normalized (total) cost for different initial values of u^{target}.

effectiveness of *ATUC* regarding the total cost, also including the periods of sub-optimal operation until convergence.

Figure 5 illustrates the results. Similarly to Section 6.3, we itemize the cost as energy cost and SLA violation penalty cost (node crash, VM relocation, node overload). For each architecture, cost is normalized w.r.t. the cost achieved by setting the initial value of $u^{target} = 0.7$.

We observe that a low initial value (0.5) results in the highest cost, for both platforms. This is expected, as in this case *ATUC* (i) requires the longest time to recover from this sub-optimal initial target utilization value, and (ii) during this time the estimated target CPU utilization of nodes remains lower than needed to achieve overload-safe operation for the architecture and workload at hand, hence the policy activates a higher number of nodes, which consume more energy. However, even in this case the total cost is only 2.98% and 2.46% higher compared with the baseline initial value for the Intel and ARM nodes, respectively. For all other initial values, we observe that the cost is similar, for both platforms. These results confirm that *ATUC* manages to recover not only quickly but also cost-effectively from adverse values for the initial setting of u^{target}.

6.5 Comparison of *ATUC* with state-of-the-art policies

In this section, we compare *ATUC* with *XM-VFS* and *LrMmt* in terms of energy and cost efficiency. We use an optimal static target utilization for *XM-VFS* and *LrMmt*, namely 0.8 and 0.7 for the Intel and ARM platform, respectively. We focus on evaluating whether *ATUC* can match the efficiency of these policies, which need in advance characterization of the workload to derive the optimal static target utilization (which is quite unrealistic for several real-world settings). The performance of these policies is highly dependent on the quantification of an appropriate static target utilization for the specific workload at hand. If the workload is not characterized in advance, or if it is characterized on a non-representative sample, performance is significantly penalized, even for small deviations of the target node utilization from the statically optimal value. For example for the workload in our evaluation, *XM-VFS* may introduce 14.65%

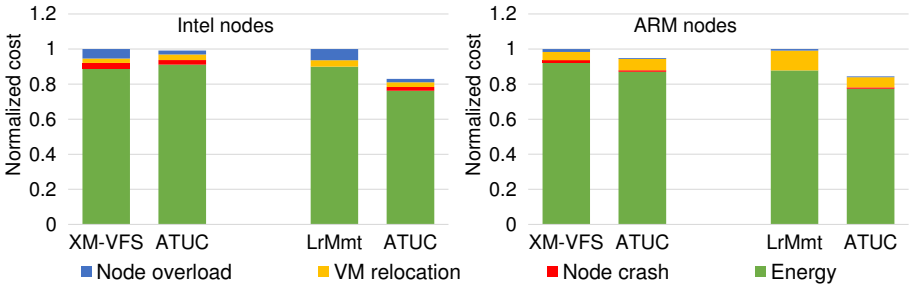

Fig. 6: Cost comparison of *ATUC* with *LrMmt* and *XM-VFS*.

higher cost on Intel nodes for a static target utilization of 0.65 (instead of the optimal 0.8).

Thanks to the effective adaptation employed by *ATUC*, the initial u^{target} value does not significantly affect cost (Section 6.4), therefore we set it at 0.7 for all these experiments. Figure 6 presents the results of the comparison between *ATUC* and the two state-of-the-art policies for the Intel and ARM platforms. Once again, we itemize the cost incurred due to energy consumption and the different sources of SLA violation penalties. All costs are normalized w.r.t. the total cost of the policy we compare against on the corresponding platform.

First, we observe that *ATUC* is not inferior to policies requiring in advance knowledge of workload characteristics. Instead, it manages to outperform them in all cases, even compared with *XM-VFS* which also exploits reduced CPU voltage margins. Specifically, *ATUC* reduces the SLA violation penalties due to overloads at the expense of a slightly higher energy cost compared against *XM-VFS* for the Intel platform, achieving 0.85% cost gains against a policy that uses an optimal static target utilization for the current workload. In the case of the ARM platform, *ATUC* achieves energy gains over *XM-VFS* without increasing the cost due to node overloading, resulting in 5.1% net cost gains. Moreover, *ATUC* manages to reduce the cost of crashes (red area) by 19.65% and 29.52% compared with *XM-VFS* for the Intel and ARM platform, respectively.

Comparing *ATUC* with *LrMmt*, we observe that *ATUC* introduces lower SLA violation costs due to VM relocation and node overloading, as it manages to adapt to the dynamic workload requirements. There are also energy gains against *LrMmt*, however, because *LrMmt* does not exploit configuration at reduced margins, unlike *ATUC* and *XM-VFS*, these energy gains come mainly from operation at reduced margins. The cost gains of *ATUC* against *LrMmt* are 17.06% and 15.61% for the Intel and ARM platform, respectively.

In summary *ATUC*, with zero-knowledge of the workload, manages to outperform *XM-VFS* and *LrMmt*, which use a statistically optimal target utilization, obtained via offline pre-characterization of the workload. Note that the feature of *ATUC* that contributes most to achieving these results is the estimation of the correlation between collocated workloads. If the correlation alone is over-

looked, the operating cost achieved by *ATUC* is 24.52% higher, despite carefully selecting a static safety slack value.

7 Related Work

Making educated decisions on the trade-off between energy cost and potential SLA violation penalties is a challenging, yet necessary undertaking towards cost reduction for cloud infrastructure providers. VM consolidation is a commonly used technique that tries to consolidate VMs to fewer nodes (without triggering SLA violations) to create opportunities to switch nodes off. The authors in [12,17] propose heuristics to increase the energy gains when applying VM consolidation.

Handling the aforementioned trade-off is further complicated by the dynamic variations of typical cloud workloads, which – if not guarded-against through overallocation of resources – may affect the quality of service (QoS) received by the end-users, increasing the SLA violation cost. The work in [10] uses Reinforcement Learning (RL) to adapt to a dynamic environment, while reducing the energy consumption and preserving the performance of the system. Work in [22] introduces an adaptive, multi-threshold framework to categorize the nodes according to their load to drive VM management considering both energy consumption and SLA violations. Authors in [14] discuss an approach to satisfy the maximum response time requirements in multi-tier applications for Cloud, detecting and resolving bottlenecks and predicting the optimal configuration. The study in [20] uses fuzzy logic to estimate a lower and upper utilization threshold of the nodes, which is used to characterize nodes as underloaded or overloaded, and to optimize the VM placement in order to reduce energy consumption and SLA violation penalties. The authors in [5] deal with the dynamic resource requirements of VMs, proposing heuristics for VM allocation and methods for estimating a target utilization threshold for the nodes. Authors in [21] deal with the dynamic workload problem, proposing two regression-based algorithms to estimate the upper utilization threshold of the nodes and detect overloads, combined with an algorithm that selects VMs from overloaded nodes, minimizing the VM migration time. In contrast with [5] and [21], our policy – apart from detecting node overloads – considers the target utilization of the nodes for the mapping of VMs to nodes, both for new VMs and for relocated ones. Moreover, unlike policies in [5] and [21], *ATUC* does not employ arbitrary safety parameters, which directly affect the energy vs. SLA penalty costs trade-off.

Apart from VM consolidation, works that combine VM management with DVFS can exploit opportunities for additional energy gains, as DVFS allows processors to operate at more power-efficient states. The work in [16] discusses the importance of combining VM consolidation with DVFS in datacentres. The study in [18] introduces an algorithm that employs an adaptive threshold estimation to improve the performance through task replication, while saving energy through DVS. Authors in [3] extend the policies in [5]. They combine VM consolidation with DVFS, and achieve significant energy gains compared with [5]. Our previous work in [15] combines VM consolidation and DVFS with CPU configu-

ration at reduced voltage margins, aggressively reducing energy at the expense of an increased probability of crashes. However, similarly to *LrMmt*, the best performing policy in [3], they exploit a static CPU utilization target estimated through offline workload characterization. Our work combines adaptivity, VM consolidation, DVFS and CPU operation at reduced margins in a cost-driven profit maximization approach. It also exploits online analysis of past VM behavior, rather than offline pre-characterization, to reduce the risk of costly, hard to predict and mitigate node overloads.

8 Conclusions

In this paper, we introduced an adaptive VM scheduling and node configuration policy (*ATUC*). Our policy identifies profitable points in the multiparametric space of VM-node mappings and node CPU configurations, using exclusively information collected at execution time. We combine a scalable, architecture-agnostic, statistical model that estimates a target CPU utilization for each node independently, considering VM allocations to the node and runtime information on past CPU usage of the VMs, with VM consolidation, VFS and CPU operation at reduced voltage margins. We evaluated our approach through simulations, using real datacentre workload traces and hardware parameters derived from real Intel- and ARM-based systems. We found that *ATUC* outperforms, in all cases, two state-of-the-art policies (*XM-VFS* and *LrMmt*) that assume *a priori* workload knowledge and use offline characterization to derive an optimal static target utilization for the workload at hand.

References

1. Amazon EC2 pricing. https://aws.amazon.com/ec2/pricing/
2. Eletric Power Monthly. https://www.eia.gov/electricity/monthly/
3. Arroba, P., Moya, J.M., Ayala, J.L., Buyya, R.: Dynamic Voltage and Frequency Scaling-aware dynamic consolidation of virtual machines for energy efficient cloud data centers. Concurrency and Computation: Practice and Experience **29**(10), e4067 (2017)
4. Beloglazov, A., Abawajy, J., Buyya, R.: Energy-aware resource allocation heuristics for efficient management of data centers for cloud computing. Future generation computer systems **28**(5), 755–768 (2012)
5. Beloglazov, A., Buyya, R.: 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 **24**(13), 1397–1420 (2012)
6. Calheiros, R.N., Ranjan, R., Beloglazov, A., De Rose, C.A., Buyya, R.: CloudSim: a toolkit for modeling and simulation of cloud computing environments and evaluation of resource provisioning algorithms. Software: Practice and experience **41**(1), 23–50 (2011)
7. Cao, Z., Dong, S.: An Energy-aware Heuristic Framework for Virtual Machine Consolidation in Cloud Computing. The Journal of Supercomputing **69**(1), 429–451 (2014)

8. Dayarathna, M., Wen, Y., Fan, R.: Data Center Energy Consumption Modeling: A Survey. IEEE Communications Surveys & Tutorials **18**(1), 732–794 (2016)
9. Engbers, N., Taen, E.: Green Data Net. Report to IT Room INFRA. European Commission. FP7 ICT 2013.6.2;2014 (2016)
10. Farahnakian, F., Liljeberg, P., Plosila, J.: Energy-Efficient Virtual Machines Consolidation in Cloud Data Centers using Reinforcement Learning. In: 2014 22nd Euromicro International Conference on Parallel, Distributed, and Network-Based Processing. pp. 500–507 (Feb 2014). https://doi.org/10.1109/PDP.2014.109
11. Garg, S.K., Gopalaiyengar, S.K., Buyya, R.: SLA-based Resource Provisioning for Heterogeneous Workloads in a Virtualized Cloud Datacenter. In: International conference on Algorithms and architectures for parallel processing. pp. 371–384. Springer (2011)
12. Ghribi, C., Hadji, M., Zeghlache, D.: Energy Efficient VM Scheduling for Cloud Data Centers: Exact allocation and migration algorithms. In: 2013 13th IEEE/ACM International Symposium on Cluster, Cloud, and Grid Computing. pp. 671–678 (May 2013). https://doi.org/10.1109/CCGrid.2013.89
13. Herbert, S., Marculescu, D.: Analysis of Dynamic Voltage/Frequency Scaling in Chip-Multiprocessors. In: 2007 ACM/IEEE International Symposium on Low Power Electronics and Design (ISLPED). pp. 38–43 (Aug 2007). https://doi.org/10.1145/1283780.1283790
14. Iqbal, W., Dailey, M.N., Carrera, D., Janecek, P.: Adaptive resource provisioning for read intensive multi-tier applications in the cloud. Future Generation Computer Systems **27**(6), 871–879 (2011)
15. Kalogirou, C., Koutsovasilis, P., Antonopoulos, C.D., Bellas, N., Lalis, S., Venugopal, S., Pinto, C.: Exploiting CPU Voltage Margins to Increase the Profit of Cloud Infrastructure Providers. In: 2019 19th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (CCGRID). pp. 302–311. IEEE (2019)
16. von Laszewski, G., Wang, L., Younge, A.J., He, X.: Power-aware scheduling of virtual machines in DVFS-enabled clusters. In: 2009 IEEE International Conference on Cluster Computing and Workshops. pp. 1–10 (Aug 2009). https://doi.org/10.1109/CLUSTR.2009.5289182
17. Lee, Y.C., Zomaya, A.Y.: Energy efficient utilization of resources in cloud computing systems. The Journal of Supercomputing **60**(2), 268–280 (2012)
18. Liu, W., Du, W., Chen, J., Wang, W., Zeng, G.: Adaptive energy-efficient scheduling algorithm for parallel tasks on homogeneous clusters. Journal of Network and Computer Applications **41**, 101–113 (2014)
19. Reiss, C., Wilkes, J., Hellerstein, J.L.: Google cluster-usage traces: format + schema. Technical report, Google Inc., Mountain View, CA, USA (Nov 2011), revised 2014-11-17 for version 2.1. Posted at https://github.com/google/cluster-data
20. Salimian, L., Esfahani, F.S., Nadimi-Shahraki, M.H.: An adaptive fuzzy threshold-based approach for energy and performance efficient consolidation of virtual machines. Computing **98**(6), 641–660 (2016)
21. Yadav, R., Zhang, W., Kaiwartya, O., Singh, P.R., Elgendy, I.A., Tian, Y.C.: Adaptive Energy-Aware Algorithms for Minimizing Energy Consumption and SLA Violation in Cloud Computing. IEEE Access **6**, 55923–55936 (2018)
22. Zhou, Z., Abawajy, J., Chowdhury, M., Hu, Z., Li, K., Cheng, H., Alelaiwi, A.A., Li, F.: Minimizing SLA violation and power consumption in Cloud data centers using adaptive energy-aware algorithms. Future Generation Computer Systems **86**, 836–850 (2018)