An Energy-Efficient VM Prediction and Migration Framework for Overcommitted Clouds

Mehiar Dabbagh, Bechir Hamdaoui, Mohsen Guizani[†] and Ammar Rayes[‡]
Oregon State University, Corvallis, dabbaghm,hamdaoui@onid.orst.edu
[†] University of Idaho, Moscow, mguizani@ieee.org
[‡] Cisco Systems, San Jose, rayes@cisco.com

***Abstract*—We propose an integrated, energy-efficient, resource allocation framework for overcommitted clouds. The framework makes great energy savings by 1) minimizing Physical Machine (PM) overload occurrences via VM resource usage monitoring and prediction, and 2) reducing the number of active PMs via efficient VM migration and placement. Using real Google data consisting of a 29-day traces collected from a cluster containing more than 12K PMs, we show that our proposed framework outperforms existing overload avoidance techniques and prior VM migration strategies by reducing the number of unpredicted overloads, minimizing migration overhead, increasing resource utilization, and reducing cloud energy consumption.**

***Index Terms*—Energy efficiency, VM migration, workload prediction, cloud computing.**

I. Introduction

Reducing the energy consumption of datacenters has received a great attention from the academia and the industry recently [1, 2]. Recent studies indicate that datacenter servers operate, most of the time, at between 10% and 50% of their maximal utilizations. These same studies, on the other hand, also show that servers that are kept ON but are idle or lightly utilized consume significant amounts of energy, due to the fact that an idle ON server consumes more than 50% of its peak power [3, 4]. It can therefore be concluded that in order to minimize energy consumption of datacenters, one needs to consolidate cloud workloads into as few servers as possible.

Upon receiving a client request, the cloud scheduler creates a virtual machine (VM), allocates to it the exact amounts of CPU and memory resources requested by the client, and assigns it to one of the cluster's physical machines (PMs). In current cloud resource allocation methods, these allocated resources are reserved for the entire lifetime of the VM and are released only when the VM completes. A key question, constituting the basis for our work motivation, that arises now is to see whether the VMs do utilize their requested/reserved resources fully, and if not, what percentage of the reserved resources is actually being utilized by the VMs. To answer this question, we conduct some measurements on real Google traces[1] [5] and show in Fig. 1 a one-day snapshot of this percentage. Observe that only about 35% of the VMs' requested CPU resources and only about 55% of the VMs' requested memory resources are actually utilized by the VMs. Our study clearly indicates that cloud resources tend to be overly reserved, leading to substantial CPU and memory resource wastage. Two main reasons are behind this over-reservation tendency: First, cloud clients usually do not know the exact amounts of resources their applications would need, so they tend to overestimate them in order to guarantee a safe execution. Second, due to and depending on the nature of the applications hosted on the PMs, the level of utilization of the requested resources may change over time and may even rarely reach its peak, making it impossible for the VM to use the full amount of its requested resources.

This work was supported in part by NSF CAREER award CNS-0846044.

[1]It is important to mention that Google, as reported in the traces, allocates containers instead of full VMs when handling task requests. However, our framework remains applicable regardless of whether tasks are handled by containers or by full virtualization. Throughout this work, we refer to the task requests submitted to the Google cluster by VM requests.

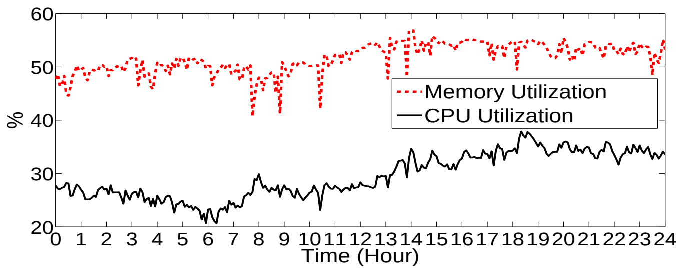

Fig. 1. One-day snapshot of Google cluster traces: percentage of utilized resources among all reserved resources.

Resource overcommitment [6, 7] is a technique that has been recognized as a potential solution for addressing the above-mentioned wastage issues. It essentially consists of allocating VM resources to PMs in excess of their actual capacities, expecting that these actual capacities will not be exceeded since VMs are not likely to utilize their reserved resources fully. Therefore, it has a great potential for saving energy in cloud centers, as VMs can now be hosted on fewer ON PMs.

Resource overcommitment creates, however, a new problem, PM overloading, which occurs when the aggregate resources demands of the VMs scheduled on some PM does indeed exceed the PM's capacity. When this occurs, some or all of the VMs running on the overloaded PM will experience performance degradation (some VMs may even crash), possibly leading to violations of service-level agreements (SLAs) between the cloud and its clients. Although VM migration [8] can be used to handle PM overloads, where some of the VMs hosted by the overloaded PM are moved to other under-utilized or idle PMs, it raises two key challenges/questions:

i) When should VMs be migrated?

ii) Which VMs should be migrated and which PMs these VMs should be migrated to?

This paper addresses these two challenges. It proposes an integrated resource allocation framework that improves resource utilization, reduces energy consumption, and avoids, as much as possible, SLA violations in cloud datacenters. More specifically, our proposed framework:

- predicts future resource utilizations of scheduled VMs, and uses these predictions to make efficient cloud resource overcommitment decisions to increase utilization.
- predicts PM overload incidents and triggers VM migrations before overloads occur to avoid SLA violations.
- performs energy-efficient VM migration by determining which VMs to migrate and which PMs need to host the migrated VMs such that the migration energy overheads and the number of active PMs are minimized.

The effectiveness of our techniques is evaluated and compared against existing ones by means of real Google traces [5] collected from a heterogeneous cluster made up of more than 12K PMs.

The remainder of the paper is organized as follows. Section II discusses prior work. Section III provides an overview of the proposed framework. Section IV presents our proposed prediction methods. Section V presents our formulation of the VM migration problem, and Section VI presents a heuristic for solving it. Section VII evaluates the effectiveness of the proposed framework. Finally, Section VIII concludes the paper and provides directions for future work.

II. Related Work

Smart cluster and host selection [9, 10], energy routing enhancement [11, 12] and packet-forwarding optimization [13, 14] are some of the techniques that were proposed to minimize the energy consumption of cloud centers. Our framework complements these general techniques and uses overbooking/overcommitment to minimize the number of ON PMs. In our previous work [15], we proposed a technique that reduces energy consumption of datacenters by limiting/controlling the number of active PMs while still supporting the demanded VM workload. It does so by exploiting predicted future resource demands to reduce the number of active/ON PMs, and scheduling newly submitted VMs on the selected PMs to increase resource utilization and reduce energy consumption. In this paper, our focus is on PM overcommitment techniques that increase resource utilization (and thus also reduce energy consumption) by essentially assigning VMs to a PM in excess of its real capacity, anticipating that each assigned VM will only utilize a part of its requested resources. Our proposed framework basically consists of first determining the amount of PM resources that can be overcommitted by monitoring and predicting the future resource demands of admitted VMs, and then handling PM overloading problems, which occur when the aggregated amount of resources utilized by the VMs exceeds a PM's capacity. Our framework handles PM overloads by predicting them before occurring and migrating VMs from overloaded PMs to other under-utilized or idle PMs whenever an overload occurs or is predicted to occur. We review next how our framework differs from prior overload avoidance and VM migration techniques in overcommitted clouds.

A. Overload Avoidance Techniques

One approach proposed in [16–18] to handle PM overloads consists essentially of triggering VM migrations upon detecting overloads. This stops VMs from contending over the limited resources, but can clearly result in some SLA violations. To address this limitation, the techniques presented in [19–25], which are referred to as threshold-based techniques, propose to trigger migrations as soon as the PM's utilization exceeds a certain threshold, but before an overload actually takes place. The assumption here is that there is a high chance that an overload occurs when the PM's utilization exceeds the set threshold. The threshold could be set statically as in VMware [19] and as in [20–22], where the threshold is typically set to 90% utilization; or dynamically as in [23–25], where it is tuned for each PM based on how fluctuating the PM's workload is. The higher the PM's utilization fluctuations, the lower the PM's selected threshold and vice versa. Although threshold-based techniques reduce overloads, they put a limitation on the utilization gains that can be achieved as they leave a certain unutilized slack for each PM. Furthermore, these techniques can trigger many unnecessary migrations as exceeding the set threshold does not necessary mean that an overload is going to happen.

To tackle these limitations, we propose a prediction-based overload-avoidance technique that predicts future resource demands for each scheduled VM so that overloads can be foreseen and migrations can be triggered ahead of time. These predictions also help us decide where to place the newly submitted VMs so that the utilization gains due to overcommitment are increased while avoiding overloads as much as possible.

There have been previous attempts to predict the resource demands of cloud applications [26–28]. Fourier transformation [26] and SVM predictors [27] were used to extract periodic patterns from the traces of certain cloud applications. The extracted patterns were used later to predict the resource demands for those applications. MapReduce jobs were profiled in [28] with the aim of predicting the resource demands for newly submitted instances of those jobs. Our prediction module differs from the previously proposed *offline* predictive-based techniques [26–28] in that it uses a customized adaptive Wiener filter [29, 30] that learns and predicts the resource demands of the clients' scheduled VMs *online* without requiring any prior knowledge about the hosted VMs. The closest work to our proposed predictive technique is the work in [31], where the authors developed an *online* scheme called Fast Up Slow Down (FUSD) that predicts the future demands of the scheduled VMs and that triggers migrations to avoid overloads. The FUSD scheme [31] and the threshold-based overload avoidance techniques [20–22] are used as benchmarks where we show in our evaluations that our proposed prediction technique not only reduces the number of unpredicted overloads, but also achieves greater energy savings when compared to those benchmark techniques.

B. VM Migration Techniques

Live migration is now available in VMware [19] and Xen [32] where the downtime that the clients experience due to the migration process is extremely low and ranges between tens of milliseconds to a second [33]. Despite the maturity of the VM migration technology, it is still challenging to find efficient resource management strategies that decide what VMs to migrate and what PMs should be the destination for each migration when an overload is encountered.

Largest First heuristics [34–37] addresses this challenge by moving VMs with the largest resource demands to the PMs with the largest resource slacks while trying to minimize the number of needed migrations. Several enhancements were added to this heuristic in [38, 39] with the aim of placing VMs that often communicate with each other close to one another. The main limitation of all the previous heuristics is that they completely ignore the energy overhead associated with migrations.

The authors in [40, 41] proposed migration policies that take migration cost into account. One of the drawbacks of the approach presented in [40] lies in its limitation to VMs with single resource (CPU). Sandpiper [41] is a multi-resource heuristic that aims at moving the largest amount of resources from the overloaded PM with the least cost. However, both works of [40, 41] share a common limitation as they completely ignore the power state of the PMs in the cluster. Although these approaches may move VMs that have low migration cost, there may be no already-ON PMs that have enough slack to host the migrated VMs. This forces turning PMs from sleep to ON to host the moved VMs, which comes with a high energy cost [15, 42] and increases the number of ON machines in the cloud cluster, leading to larger energy consumption.

Unlike previous works, our proposed framework has an energy-aware migration module that makes migration decisions that minimize the total migration energy overhead, which is made up of the energy spent to move VMs and that to switch PMs ON to host the migrated VMs. Our experimental studies presented in Section VII show that our proposed migration method achieves higher energy savings and consolidates the workload in fewer PMs when compared to both the Largest First and Sandpiper heuristics, discussed above.

III. Proposed Framework

Our proposed framework is suitable for heterogeneous cloud clusters whose PMs may or may not have the same resource capacities. Although our framework is extendable to any number of resources, in this paper, we consider two resources: CPU and memory. Thus, a PM j can be represented by $[C^j_{cpu}, C^j_{mem}]$, where C^j_{cpu} and C^j_{mem} are the PM's CPU and memory capacities. Throughout, let P be the set of all PMs in the cloud cluster. Recall that a client may, at any time, submit a new VM request, say VM i, represented by $[R^i_{cpu}, R^i_{mem}]$ where R^i_{cpu} and R^i_{mem} are the requested amounts of CPU and memory. Whenever the client no longer needs the requested resources, it submits a VM release request. Throughout, let V be the set of all VMs hosted by the cluster.

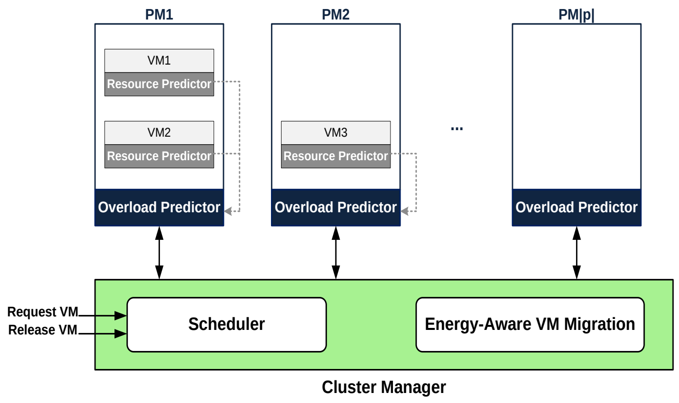

Fig. 2. Flow chart of the proposed framework

In this section, we provide a brief overview of the different components of the proposed framework so as to have a global picture of the entire framework before delving into the details. As shown in Fig. 2, our framework has the following modules:

A. Resource Predictor

A separate Resource Predictor module is dedicated to each VM scheduled on the cluster. The Resource Predictor module for VM i consists of two predictors (one for CPU and one for memory) that monitor and collect the VM's CPU and memory usage traces, and use them, along with other VM parameter sets (to be learned online from the VM's resource demands behaviors), to predict the VM's future CPU and memory demands, P^i_{cpu} and P^i_{mem}. These predictions are calculated for the coming τ period and are done periodically at the end of each period. Detailed description of how predictors work and how these parameters are updated are given in Section IV.

B. Overload Predictor

Each PM in the cluster is dedicated an Overload Predictor module. The Overload Predictor module assigned for PM j fetches the predicted CPU and memory demands for all the VMs that are hosted on that PM and calculates PM j's predicted aggregate CPU and memory demands, U^j_{cpu} and U^j_{mem} respectively, as follows:

$$U^j_{cpu} = \sum_{i \in V:\theta(i)=j} P^i_{cpu} \text{ and } U^j_{mem} = \sum_{i \in V:\theta(i)=j} P^i_{mem}$$

where $\theta : V \to P$ is the VM-PM mapping function, with $\theta(i) = j$ meaning that VM i is hosted on PM j. The module then compares the calculated aggregate CPU and memory demands, U^j_{cpu} and U^j_{mem}, with PM j's supported CPU and memory capacity, C^j_{cpu} and C^j_{mem}. If $U^j_{cpu} > C^j_{cpu}$ or $U^j_{mem} > C^j_{mem}$, then the Overload Predictor notifies the

Cluster Manager that PM j is expected to have an overload in the coming period. The Overload Predictor also forwards to the Cluster Manager the predicted CPU and memory demands, P^i_{cpu} and P^i_{mem}, for each VM i hosted on PM j. These predictions will be used by the Energy-Aware Migration module to decide what VM(s) to keep, what VM(s) to migrate and to where the migrated VM(s) should be moved such that the predicted overload is avoided in future. If no overload is predicted on PM j, then the Overload Predictor module forwards to the Cluster Manager the predicted CPU slack and memory slack for PM j, referred to by S^j_{cpu} and S^j_{mem} respectively, which are calculated as follows:

$$S^j_{cpu} = C^j_{cpu} - U^j_{cpu} \text{ and } S^j_{mem} = C^j_{mem} - U^j_{mem} \qquad (1)$$

The CPU and memory slacks on PM j need to be known to the Cluster Manager as it may decide to place a VM (a newly submitted VM or a migrated VM) on PM j and thus the Cluster Manager needs to make sure that enough slacks are available on the PM to support the resource requirements of the newly placed VM.

As expected with any prediction framework, it is also possible that our predictors fail to predict an overload. We refer to such incidents as *unpredicted overloads*, which will be eventually detected when they occur as the Overload Predictor also tracks how much resources all the VMs scheduled on the PM are actually using over time. For any predicted PM overload, VM migration will be performed before the overload actually occurs, thus avoiding it. But for each unpredicted PM overload, VM migration will be performed upon its detection. All VM migrations are handled by the Energy-Aware VM Migration sub module in the Cluster Manager.

C. Cluster Manager

This module is connected to the Overload Predictor modules of each PM in the cluster and is made up of two sub modules:

1) Energy-Aware VM Migration: Let O_{pm} be the set of all PMs that are predicted to have an overload. This sub module determines which VM(s) among those hosted on the PMs in O_{pm} need to be migrated so as to keep the predicted aggregate CPU and memory demands below the PM's capacity. To make efficient decisions, the module needs to know the energy costs for moving each VM among those hosted on the PMs in O_{pm}. We use the notation m_i to refer to the cost (in Joules) for moving the i^{th} VM. This module also determines which PM each migrating VM needs to migrate to. Such a PM must have enough CPU and memory slack to accommodate the migrated VM(s), and thus the module needs to know the available resource slacks on all the PMs in the cluster that are not predicted to have an overload. These are provided to the module as input (through the Overload Prediction module) in addition to the ON-sleep states of the PMs, γ, where the state function $\gamma(j)$ returns PM j's power state (ON or sleep) prior to migration. After performing the required migrations, the VM-PM mapping θ and the ON-sleep PM state γ get updated. Details on how the the migration problem is formulated and how it is solved by our module are provided in Sections V and VI.

2) Scheduler: This sub module decides where to place newly submitted VMs and also handles VM release events.

The new VM placements are handled with two objectives in mind: saving energy and minimizing the PM overload occurrence probability. When the client requests a VM, the Scheduler places the VM request on a sleeping PM only if no ON PM is predicted to have enough CPU slack (S_{cpu}) and enough memory slack (S_{mem}) to provide the requested CPU and memory resources of the submitted VM request. The sleeping PM with the largest capacity ($C_{cpu} \times C_{mem}$) in that case is switched ON and is selected to host the submitted VM request. On the other hand, if multiple ON PMs have enough predicted CPU and memory slacks to host the submitted VM request, then the *predicted slack metric* (defined for a PM j as $S^j_{cpu\times mem} = S^j_{cpu} \times S^j_{mem}$) is calculated for each one of those PMs. The ON PM with the largest predicted slack metric among those that can fit the VM request is selected to host the submitted VM.

The intuition behind our placement policy is as follows: It is better to host a newly submitted VM request on an ON PM, so as to avoid wakening up asleep machines. This saves energy. If there is no option but to place the VM request on a sleeping PM, then the request is placed on the sleeping PM that has the largest capacity as this PM can fit a larger number of VM requests that might be received next, which reduces the chances of waking up another sleeping PM in future. On the other hand, if multiple ON PMs can fit the submitted VM request, then the one with the largest predicted slack is picked to host the submitted VM request so as to decrease the chances of having an overload after that placement. The product of the CPU and memory resources is used to combine the server's multiple resources into a single sorting metric. This metric is used to decide which sleeping PM has the largest capacity or which ON PM has the largest predicted slack when comparing two PMs.[2]

A Resource Predictor module is constructed for the new VM after assigning it to a PM. The Resource Predictor module consists of two predictors (one for CPU and one for memory) and will be used to monitor the resource demands of the VM and to make future demand predictions, as described earlier.

Upon receiving a VM release from the client, the Scheduler releases the VM's allocated CPU and memory resources, frees all system parameters associated with the VM (e.g., predictors), and updates the aggregate CPU and memory predictions of the hosting PM accordingly. The PM is switched to sleep to save energy if it becomes vacant after releasing the VM.

[2]It is worth mentioning that there exists metrics other than the product that can be used to combine the multiple resources. The summation of the CPU and memory resources can be used for example as a combining metric. In our framework, we selected the product metric over the summation metric as experiments in our prior work [15] showed that for the Google traces, the product metric makes slightly more compact VM placements when compared to the summation metric. The other metrics mentioned in [43] such as the Dot Product and the Norm-Based Greedy were not considered in our framework due to their high computational overhead.

IV. VM Resource Predictor

We explain in this section how a predictor for a scheduled VM predicts its future resource demands in the coming τ minutes, where the term resource will be used to refer to either CPU or memory. In our framework, we choose to use the Wiener filter prediction approach [29, 30] for several reasons. First, it is simple and intuitive, as the predicted utilization is a weighted sum of the recently observed utilization samples. Second, prediction weights can easily be updated without requiring heavy calculations or large storage space. Finally, it performs well on real traces as will be seen later.

Let n be the time at which resource predictions for a VM need to be made. The following notations will be used throughout:

- $z[n-i]$: is the VM's average resource utilization during period $[n-(i+1)\tau, n-i\tau]$ minutes.
- $d[n]$: is the VM's actual average resource utilization during period $[n, n+\tau]$.
- $\hat{d}[n]$: is the VM's predicted average resource utilization during period $[n, n+\tau]$.

Wiener filters predict resource utilizations while assuming wide-sense stationarity of $z[n]$. The predicted average resource utilization, $\hat{d}[n]$, is a weighted average over the L most recently observed utilization samples; i.e., $\hat{d}[n] = \sum_{i=0}^{L-1} w_i z[n-i]$, where w_i is the i^{th} sample weight. The prediction error, $e[n]$, is then the difference between the actual and predicted utilizations; i.e., $e[n] = d[n] - \hat{d}[n] = d[n] - \sum_{i=0}^{L-1} w_i z[n-i]$. The objective is to find the weights that minimize the Mean Squared Error (MSE) of the training data, where $MSE = E[\,e^2[n]\,]$. Differentiating MSE with respect to w_k and setting this derivative to zero yields, after some algebraic simplifications, $E\big[d[n]z[n-k]\big] - \sum_{i=0}^{L-1} w_i E\big[z[n-k]z[n-i]\big] = 0$. It then follows that $r_{dz}(k) = \sum_{i=0}^{L-1} w_i r_{zz}(i-k)$ where

$$r_{dz}(k) = E\big[d[n]z[n-k]\big] \tag{2}$$

$$r_{zz}(i-k) = E\big[z[n-k]z[n-i]\big] \tag{3}$$

Similar equations expressing the other weights can also be obtained in the same way. These equations can be presented in a matrix format as $R_{dz} = R_{zz}W$, where

$$R_{zz} = \begin{bmatrix} r_{zz}(0) & r_{zz}(1) & \dots & r_{zz}(L-1) \\ r_{zz}(1) & r_{zz}(0) & \dots & r_{zz}(L-2) \\ \vdots & \vdots & \ddots & \vdots \\ r_{zz}(L-1) & r_{zz}(L-2) & \dots & r_{zz}(0) \end{bmatrix}$$

$$W = \begin{bmatrix} w_0 & w_1 & \dots & w_{L-1} \end{bmatrix}^T$$

$$R_{dz} = \begin{bmatrix} r_{dz}(0) & r_{dz}(1) & \dots & r_{dz}(L-1) \end{bmatrix}^T$$

Given R_{zz} and R_{dz}, the weights can then be calculated as:

$$W = R_{zz}^{-1} R_{dz} \tag{4}$$

The elements of R_{zz} are calculated using the unbiased correlation estimation as:

$$r_{zz}(i) = \frac{1}{N-i} \sum_{j=0}^{N-i-1} z[j+i]z[j] \tag{5}$$

where N is the VM's number of observed samples with each sample representing an average utilization over τ minutes.

The elements of R_{dz} can also be estimated using the correlation coefficients. Since $d[n]$ represents the average resource utilization in the coming τ minutes, we can write $d[n] = z[n+1]$. Plugging the expression of $d[n]$ in Eq. (2) yields $r_{dz}(k) = E[z[n+1]z[n-k]] = r_{zz}(k+1)$, and thus $R_{dz} = \begin{bmatrix} r_{zz}(1) & r_{zz}(2) & \dots & r_{zz}(L) \end{bmatrix}^T$. The elements of R_{dz} can be calculated using Eq. (5). An MSE estimation of the weight vector follows then provided R_{dz} and R_{zz}.

a) Adaptive Updates: Note that $N = L+1$ samples need to be observed in order to calculate the predictor's weights. When the number of samples available during the early period is less than $L+1$, no prediction will be made and we assume that VMs will utilize all of their requested resources. Once the predictor observes $N = L+1$ samples, the correlations $r_{zz}(i)$ can then be calculated for all i allowing the weights to be estimated.

When $N > L+1$, the predictor adapts to new changes by observing the new utilization samples, updates the correlations, and calculates the new updated weights. This results in increasing the accuracy of the predictor over time, as the weights are to be calculated based on a larger training data. From Eq. (5), the coefficient $r_{zz}(i)$ can be written as $Sum(i)/Counter(i)$, where $Sum(i) = \sum_{j=0}^{N-i-1} z[j+i]z[j]$ and $Counter(i) = N-i$ are two aggregate variables. Now recall that every τ minutes, a new resource utilization sample $z[k]$ is observed, and hence, the aggregate variables can be updated as $Sum(i) \leftarrow Sum(i) + z[k]z[k-i]$ and $Counter(i) \leftarrow Counter(i) + 1$ and the correlation $r_{zz}(i)$ is updated again as $Sum(i)/Counter(i)$. The updated weights are then calculated using Eq. (4), which will be used to predict the VM's future resource utilizations. Note that only two variables need to be stored to calculate r_{zz} instead of storing all the previous traces, and thus the amount of storage needed to update these weights is reduced significantly.

b) Safety Margin: Our stochastic predictor may still make errors by over- or under-estimating resource utilizations. When under-estimation occurs, overcommited PMs may experience overloads, potentially leading to some performance degradation. In order to take an even more conservative approach towards reducing such occurrences, we add a safety margin. We consider an adaptive approach for setting safety margins, where margin values are adjusted based on the accuracy of the predictor. Essentially, they depend on the deviation of the predicted demands from the actual ones; i.e., the higher the deviation, the greater the safety margin. In our framework, we calculate an exponentially weighted moving average (EWMA) of the deviation, Dev, while giving higher weights to the most recent ones. Initially, we set $Dev[0] = 0$, and for later time n, we set $Dev[n] = (1-\alpha)Dev[n-1] + \alpha\left|d[n-1] - \hat{d}[n-1]\right|$, where $0 < \alpha < 1$ is the typical EWMA weight factor used to tune/adjust the weight given to most recent deviations. The predicted average utilization with safety margin, $\bar{d}[n]$, can then be calculated as $\bar{d}[n] = \hat{d}[n] + Dev[n]$. Recall that each VM requests two resources: CPU and memory. Hence, two predictions $\bar{d}^v_{cpu}[n]$ and $\bar{d}^v_{mem}[n]$ are calculated as described above for each VM v. Putting it all together, VM v's predicted average CPU and memory utilizations, P^v_{cpu} and P^v_{mem}, are calculated as

$$P^v_{cpu} = \begin{cases} R^v_{cpu} & N < L+1 \\ \min(\bar{d}^v_{cpu}[n], R^v_{cpu}) & \text{otherwise} \end{cases}$$

$$P^v_{mem} = \begin{cases} R^v_{mem} & N < L+1 \\ \min(\bar{d}^v_{mem}[n], R^v_{mem}) & \text{otherwise} \end{cases}$$

V. Energy-Aware VM Migration

VM migration must be performed when an overload is predicted in order to avoid SLA violations. Since energy consumption is our primal concern, we formulate the problem of deciding which VMs to migrate and which PMs to migrate to as an optimization problem with the objective of minimizing the migration energy overhead as described next.

Decision Variables. Let O_{vm} be the set of VMs that are currently hosted on all the PMs that are predicted to overload in O_{pm}. For each VM $i \in O_{vm}$ and each PM $j \in P$, we define a binary decision variable x_{ij} where $x_{ij} = 1$ if VM i is assigned to PM j after migration, and $x_{ij} = 0$ otherwise. Also, for each $j \in P$, we define $y_j = 1$ if at least one VM is assigned to PM j after migration, and $y_j = 0$ otherwise.

Objective Function. Our objective is to minimize VM migration energy overhead, which can be expressed as

$$\sum_{i \in O_{vm}} \sum_{j \in P} x_{ij}\, a_{ij} + \sum_{j \in P} y_j\, b_j \tag{6}$$

and is constituted of two components: *VM moving energy overhead* and *PM switching energy overhead.* VM moving energy overhead, captured by the left-hand summation term, represents the energy costs (in Joule) associated with moving VMs from overloaded PMs. The constant a_{ij} represents VM i's moving cost, and is equal to m_i when VM i is moved to a PM j different from its current PM, and equal to 0 when VM i is left on the same PM where it has already been hosted. Formally, $a_{ij} = 0$ if, before migration, $\theta(i) = j$, and $a_{ij} = m_i$ otherwise. Here m_i denotes VM i's moving energy overhead.

PM switching energy overhead, captured by the right-hand term of the objective function (Eq. (6)), represents the energy cost associated with switching PMs from sleep to ON to host the migrated VMs. The constant $b_j = 0$ if PM j has already been ON before migration (i.e., $\gamma(j)$ = ON before migration), and $b_j = E_{s \to o}$ otherwise, where $E_{s \to o}$ is the transition energy consumed when switching a PM from sleep to ON.

Constraints: The optimization problem is subject to the following constraints. One,

$$\sum_{j \in P} x_{ij} = 1 \qquad \forall i \in O_{vm}$$

dictating that every VM must be assigned to only one PM. Two,

$$\sum_{i \in O_{vm}} x_{ij}\, P^i_{cpu} \le C^j_{cpu} \qquad \forall j \in O_{pm}$$

$$\sum_{i \in O_{vm}} x_{ij}\, P^i_{mem} \le C^j_{mem} \qquad \forall j \in O_{pm}$$

which state that the predicted CPU and memory usage of the scheduled VMs on any overloaded PM must not exceed the PM's available CPU and memory capacities. Three,

$$\sum_{i \in O_{vm}} x_{ij}\, P^i_{cpu} \le S^j_{cpu} \qquad \forall j \in P \backslash O_{pm}$$

$$\sum_{i \in O_{vm}} x_{ij}\, P^i_{mem} \le S^j_{mem} \qquad \forall j \in P \backslash O_{pm}$$

where $P \backslash O_{pm}$ is the set of PMs predicted not to be overloaded. S^j_{cpu} and S^j_{mem} are the predicted CPU and memory slacks for PM j calculated using Eq.(1). Recall that some VMs will be migrated to PMs that already have some scheduled VMs, and the above constraints ensure that there will be enough resource slack to host any of the migrated VMs. Four,

$$\sum_{i \in O_{vm}} x_{ij} \le |O_{vm}|\, y_j \qquad \forall j \in P \tag{7}$$

which forces y_j to be one (i.e., PM j needs to be ON) if one or more VMs in O_{vm} will be assigned to PM j after migration.

Note that if none of the VMs in O_{vm} is assigned to PM j, then the constraint (7) can still hold even when y_j takes on the value one. In order to force y_j to be zero when no VM is assigned to PM j (i.e. PM j maintains the same power state that it had prior to migration as no VM will be migrated to it), we add the following constraint. Five,

$$1 + \sum_{i \in O_{vm}} x_{ij} > y_j \qquad \forall j \in P \tag{8}$$

Note that if one or more VMs is assigned to PM j, constraint (8) does not force $y_j = 1$ either, but constraint (7) does. Thus, constraints (7) and (8), together, imply that $y_j = 1$ if and only if one or more VMs are assigned to PM j after migration.

After solving the above problem, the optimal y_js indicate whether new PMs need to be turned ON (also reflected via the γ function), and the optimal x_{ij}s indicate whether new VM-PM mappings are needed (also reflected via the θ function).

VI. Proposed Heuristic

In the previous section, we formulated the VM migration problem as an integer linear program (ILP). The limitation of this formulation lies in its complexity, arising from the integer variables, as well as the large numbers of PMs and VMs. To overcome this complexity, we instead propose to solve this problem using the following proposed fast heuristic.

Instead of deciding where to place the VMs that are currently hosted on all the overloaded PMs, our proposed heuristic (shown in Algorithm 1) takes only one overloaded PM P_o at a time (line 1), and solves a smaller optimization problem to decide where to place the VMs that are currently hosted on the picked PM P_o. We refer to these VMs that are hosted on P_o prior to migration by O_s (line 2). Another feature adopted by our heuristic that reduces the complexity further is to consider only a set of N_{on} ON PMs and N_{sleep} asleep

PMs as destination candidates for VM migrations. The set of selected ON PMs, denoted by P_{on}, is formed by the function `PickOnPMs` in line 4. The returned PMs by this function are the ON PMs that have the largest predicted slack metric ($S_{cpu} \times S_{mem}$). The intuition here is that the PM with the largest predicted slack has higher chances for having enough space to host the VMs that need to be migrated. Furthermore, moving VMs to these PMs has the lowest probability to trigger an overload on these PMs. The set of selected PMs that are asleep is denoted by P_{sleep} and formed using the function `PickSleepPMs`. The returned PMs are the ones that are asleep and that have the largest capacity ($C_{cpu} \times C_{mem}$). Again the intuition here is that PMs with larger capacity have larger space to host the migrated VMs and hence, the lowest probability of causing an overload.

The heuristic then forms the set P_s (line 5) which is made up of the picked overloaded PM, P_o, the selected ON PMs, P_{on}, and the selected sleep PMs, P_{sleep}. An optimization problem similar to the one explained in the previous section is solved with the only exception that $P = P_s$ and $O_{vm} = O_s$. Solving the optimization problem determines which VMs in O_s need to be migrated so as to avoid PM overloads. The VMs that are assigned to one of the ON PMs are then migrated to the assigned ON PMs. As for the VMs that are assigned to one of the PMs that are in sleep, we try first to place them in any already ON PMs. To do so, all the ON PMs apart from those in P_{on} are ordered in a decreasing order of their slacks. The VMs that are assigned to the PMs in sleep are also ordered from largest to smallest. We iterate over these VMs while trying to fit them in one of the ordered ON PMs. This is done in order to make sure that no ON PMs (other than the selected PMs in N_{on}) have enough space to host the migrated VMs. If an already ON PM has enough space to host one of these VMs, then the VM is migrated to the ON PM rather than to the PM in sleep so as to avoid incurring switching energy overhead. Otherwise, the VMs are migrated to the assigned PMs that are asleep, as indicated in line 6.

As for N_{on} and N_{sleep}, these parameters affect the size of the optimization problem. The larger the values of these parameters, the higher the number of PM destination candidates, and the longer the time needed to solve the problem but the lower the total migration energy overhead. This point will be further illustrated by the experiments presented in the following section.

Algorithm 1 [θ, γ] = Proposed Heuristic(N_{on}, N_{sleep})

```
1: for each $P_o \in O_{pm}$ do
2:     $O_s = \{\forall i \in V \, s.t. \, \theta(i) = P_o\}$
3:     $P_{on} \leftarrow$ PickOnPMs($N_{on}$)
4:     $P_{sleep} \leftarrow$ PickSleepPMs($N_{sleep}$)
5:     $P_s \leftarrow P_{on} \cup P_{sleep} \cup P_o$
6:     $[\vec{x}, \vec{y}] = SolveOptimization(P_s, O_s)$
7:     Migrate VMs that should be placed on a PM $\in P_{on}$
8:     Try placing VMs that should be placed on a PM $\in P_{sleep}$ on
       any ON PM $\notin P_{on}$
9:     Update $\theta$ and $\gamma$
10: end for
```

VII. Framework Evaluation

The experiments presented in this section are based on real traces of the VM requests submitted to a Google cluster that is made up of more than 12K PMs (see [5] for further details). Since the size of the traces is huge, we limit our analysis to a chunk spanning a 24-hour period. Since the traces do not reveal the energy costs associated with moving the submitted VMs, nor do they reveal enough information that allow us to estimate the costs to move the VMs (e.g. the memory dirtying rate, the network bandwidth, etc.), we assume that the VM's moving overhead follows a Gaussian distribution with a mean $\mu = 350 \, Joule$ and a standard deviation $\delta = 100$ [44]. The selection of these numbers is based on the energy measurements reported in [44], which show that the moving overhead varies between 150 and 550 Joules for VM sizes between 250 and 1000 Mega Bytes. These moving costs include the energy consumed by the source PM, the destination PM, and the network links.

As for the power consumed by an active PM, $P(\eta)$, it increases linearly from P_{idle} to P_{peak} as its CPU utilization, $\eta = U_{cpu}/C_{cpu}$, increases from 0 to 100% [45,46]. More specifically, $P(\eta) = P_{idle} + \eta(P_{peak} - P_{idle})$, where $P_{peak} = 400$ and $P_{idle} = 200$ Watts. A sleeping PM, on the other hand, consumes $P_{sleep} = 100$ Watts. The energy consumed when switching a PM from sleep to ON is $E_{s \to o} = 4260$ Joules, and that when switching a PM from ON to sleep is $E_{o \to s} = 5510$ Joules. These numbers are based on real servers' specs [47–49].

Overload Prediction. We start our evaluations by showing in Fig. 3 the number of overloads predicted when our framework is run over the 24-hour trace period. These overload predictions are based on predicting the VM's CPU and memory demands in the coming $\tau = 5$ minutes using our proposed Wiener filter with safety margin. Although our framework works for any τ value, the selection of $\tau = 5$ is based on the fact that Google traces report resource utilization for the scheduled VMs every 5 minutes. The number of the most recent observed samples considered in prediction is $L = 6$ as our experiments showed that considering more samples would increase the calculation overhead while making negligible accuracy improvements. As for the safety margin, The EWMA weight factor α is set to 0.25 as our experiments showed that this is the value that balanced the most between reducing the under and the over estimation error. The parameters of our migration heuristic are set to $N_{on} = 1$ and $N_{sleep} = 1$. The values of the parameters that are specified here will be used throughout all the following experiments unless otherwise specified. For the sake of comparison, we also show in Fig. 3 the number of overloads that were not predicted by our predictors. Observe how low the number of unpredicted overloads is; it is actually zero with the exception of three short peaks. This proves the effectiveness of our framework vis-a-vis of predicting overloads ahead of time, thus avoiding VM performance degradation.

In order to evaluate how effective our prediction technique

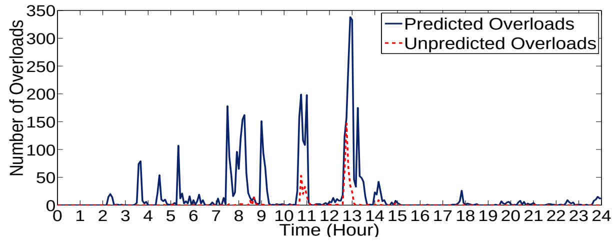

Fig. 3. Number of overloads that were predicted and that were unpredicted by our framework over time.

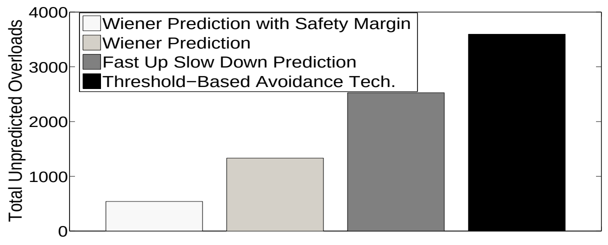

Fig. 4. Total number of unpredicted overloads during entire testing period using different overload avoidance techniques.

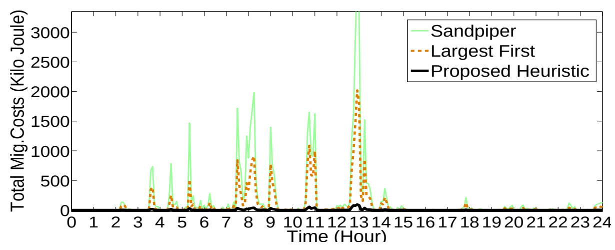

Fig. 5. Total migration energy overhead under each of the three heuristics.

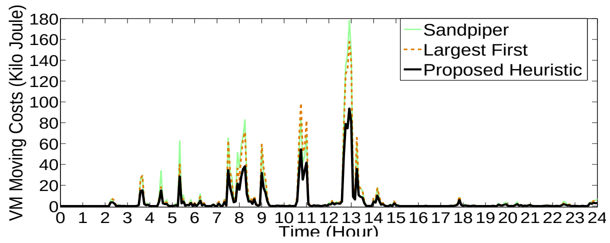

Fig. 6. VM moving energy overhead.

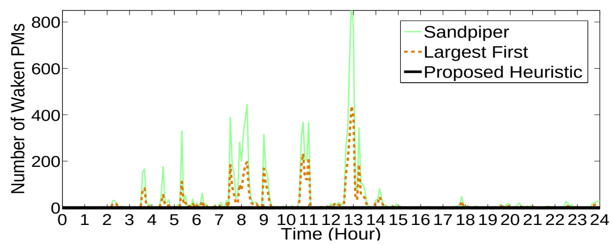

Fig. 7. The number of waken PMs to host the moved VMs.

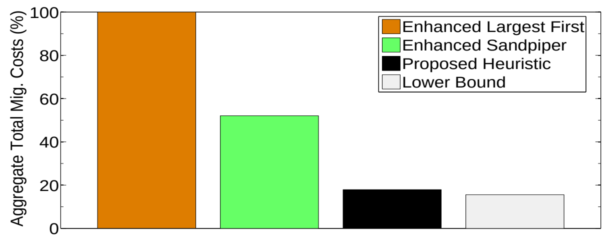

Fig. 8. Total migration overhead aggregated over the entire testing period (normalized w.r.t. Enhanced Largest First overheads).

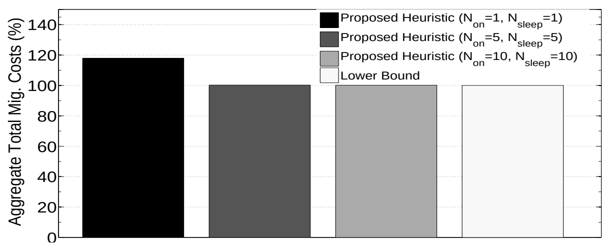

Fig. 9. Total migration overhead for the proposed migration heuristic under different values of N_{on} and N_{sleep} (normalized w.r.t. Lower Bound).

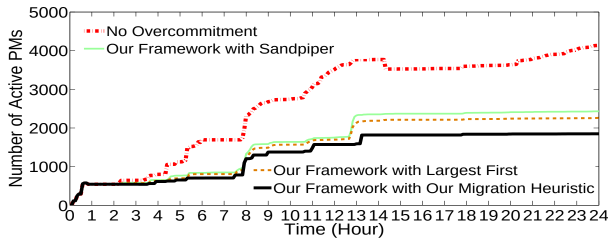

Fig. 10. Number of PMs needed to host the workload.

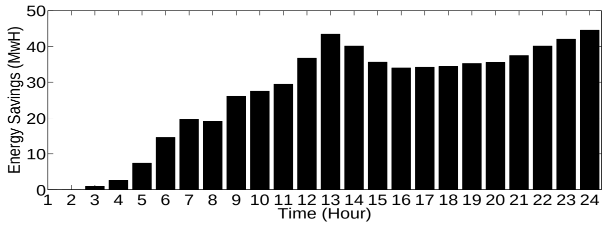

Fig. 11. Amount of energy that our allocation framework saves every hour when compared to allocation without overcommitment.

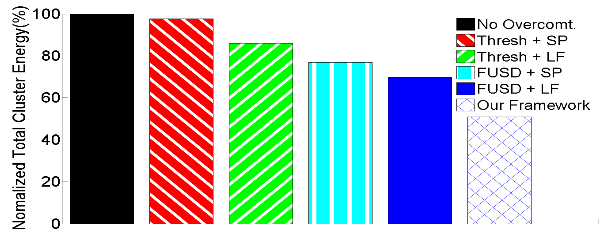

Fig. 12. Google cluster's total consumed energy during the 24-hour trace period for the different schemes.

is compared to existing overload avoidance techniques, we report in Fig. 4 the total number of unpredicted overloads during the entire 24-hour trace period for the following overload avoidance techniques: i) Fast Up Slow Down (FUSD) online prediction technique [31], and ii) the Threshold-Based Avoidance Technique [19–22] where overloads are predicted when the CPU/memory utilization of the overcommited PM exceeds 90%. We also report in Fig. 4 the total number of unpredicted overloads during the entire 24-hour trace period for our proposed Wiener filter with and without safety margin. Observe from Fig. 4 that both the FUSD and the Threshold-based avoidance technique had a larger number of unpredicted overloads compared to our proposed Wiener filter. This shows that our proposed prediction module is more effective in terms of predicting overload compared to existing techniques. Observe also that by adding a safety margin to our proposed Wiener Prediction technique, the number of unpredicted overloads was reduced further. The gap between the case with and without safety margin quantifies the benefit of adding a safety margin to our prediction module.

VM Migration Energy Overhead. Having shown the efficiency of our proposed prediction module, we now evaluate how efficient our proposed migration heuristic is when compared to existing migration strategies. In order to do that, we plot in Fig. 5 the total migration energy overhead (including both VM moving and PM switching overheads) incurred by the migration decisions of our proposed heuristic to avoid/handle the overloads reported in Fig. 3 along with the total energy overhead associated with the migration decisions of the two existing heuristics:

- Largest First [34–37]: This heuristic considers one overloaded PM at a time, and orders all its hosted VMs in a decreasing order of their resource usage. It then starts migrating VMs from the top of the ordered list, one at a time, until the PM's usage goes below its capacity.
- Sandpiper [41]: It is similar to Largest First with the exception that it uses, as a sorting metric, the VM's resource usage divided by the VM's moving energy cost.

Both of these heuristics handle multi-dimensional resources by considering the product metrics, and both select the PM with the largest slack as a destination for each migrated VM.

Observe from Fig. 5 that our proposed heuristic incurs significantly lesser total migration overhead when compared to the other two heuristics. We next analyze the two components/sources of migration energy overhead, VM moving overhead and PM switching overhead, separately, so as to shed some light on how and what each component contributes:

1) *VM Moving Energy Overhead.* Fig. 6 shows the VM moving energy overheads under each of the three heuristics throughout the 24-hour period. Observe that our heuristic has lower moving overheads. This is due to the fact that for each overloaded PM, our heuristic identifies and migrates the VMs with the least moving cost.
2) *PM Switching Energy Overhead.* Fig. 7 shows the number of PMs that were switched ON because of VM migration. Observe that the number of PMs that forced to be turned ON when hosting migrated VMs is much smaller under our heuristic than under the other two heuristics. This is because both Largest First and Sandpiper heuristics give a higher priority to the PMs with the largest slack, but without accounting for the PMs' power states.

The discussions and insights drawn above from Figs. 6 and 7 explain why our approach achieves lower total migration overhead when compared to the other heuristics as shown in Fig. 5.

Having compared the energy overhead of our migration heuristic against Largest First and Sandpiper, we now propose an enhancement for these existing heuristics so that they account for the power state of the PMs in the cluster, and evaluate our heuristic against the enhanced versions of these existing techniques. The enhancement basically makes Largest First and Sandpiper place the migrated VM on a sleeping PM only if the VM can't be fitted in any already ON PM, whereas if multiple ON PMs have enough slack to host the moved VM, then the ON PM with the largest slack is selected for placement. This is different from the original implementation of these heuristics (as described in [34–37, 41]) where the migrated VM is placed on the PM with the largest slack regardless of whether the PM is ON or asleep.

Fig. 8 shows the total migration energy overhead (including both the VM migration and the PM switching overheads) aggregated over the entire 24-hour testing period for the enhanced versions of Largest First and Sandpiper as well as for our proposed migration heuristic, normalized with respect to the aggregate overheads of the Enhanced Largest First heuristic. Results show that our heuristic incurred significantly lower migration overhead than even the enhanced version of those heuristics. The main reason why these enhanced heuristics still incurred higher overheads than our heuristic is due to the fact that they both select first what VM(s) to migrate, then they search for an ON PM to fit the VM(s) selected for migration, where they don't always succeed in finding an ON PM with enough resource slack to fit the VM(s) selected for migration and end up waking up some PMs from sleep. Furthermore, the metric that these heuristics use to combine the CPU and memory resources does not always make good VM selections for migration. Our heuristic addresses these limitations by solving a well-formulated optimization problem. Fig. 8 also shows the overheads of a lower bound of the optimal migration decisions, where the results show that our heuristic's overhead is very close to the lower bound of the optimal solution. Details on how the lower bound is calculated will be provided in the following paragraph.

Heuristic's Optimality and Execution Time. We also wanted to compare the energy overhead of our heuristic's decisions against the optimal solution obtained by solving the full optimization problem that was introduced in Section V. However, the large number of integer decision variables that are involved in the full optimization problem has prevented us from finding the optimal solution within a reasonable time. This has led us to compare our heuristic against a lower bound of the optimal solution that can be calculated quickly and efficiently.

Any time a set of PMs O_{pm} are predicted to overload, the lower bound for the migrations with the least total energy overhead to avoid the predicted overloads can be calculated as follows. For each PM in O_{pm}, the VM(s) with the least moving cost that should be removed from the considered PM to avoid the overload are identified by finding the exact solution of a two-dimensional 0-1 Knapsack problem [50]. The lower bound is then obtained by aggregating the VM moving costs of all the VMs that need to be removed from all the PMs in O_{pm}. This is a lower bound on the optimal solution as it identifies which VMs need to be migrated to avoid the predicted overloads such that the VM moving costs (the first part of the objective in Eq. 6) are minimized while completely ignoring the PM switching costs (the second part of the objective in Eq.6). This is the case since no CPU or memory resources are reserved for the removed VMs and thus no PM needs to be waken from sleep, and as the removed VMs are not contending over the resource slack that is available on the PMs that are ON in the cluster.[3]

Fig. 9 shows the lower bound energy overhead for the migrations needed to avoid all the overloads that were predicted within the 24-hour trace period. The figure also shows the total migration energy overhead (including both VM moving and PM switching overheads) for our proposed heuristic under different values of N_{on} and N_{sleep} when handling all the predicted overloads within the 24-hour trace period. Observe that for the case of $N_{on} = 1$ and $N_{sleep} = 1$, our heuristic had a migration overhead that is around 20% larger than the lower bound of the optimal solution. Observe also that after increasing the parameters of our heuristic to $N_{on} = 5$ and $N_{sleep} = 5$, the energy overhead of our heuristic becomes less than 1% larger than the lower bound of the optimal solution. This shows that the migration decision made by our heuristic are very close to the optimal case where we know for sure that the overhead of the optimal case must be larger than or equal to the lower bound. Observe also that there is no significant difference in the total overhead of our heuristic's migration decisions when increasing the parameters further to be $N_{on} = 10$ and $N_{sleep} = 10$.

We also show in the second column of Table I how much time on average it took our heuristic to make migration decisions any time a set of PMs were predicted to overload within the 24-hour trace period under the different parameters. The execution time is based on running a Matlab code for our heuristic on an x-86 platform that has 2 sockets, each socket has 8 cores, each core has 2 threads with a CPU frequency of 2.6 Ghz and a 62 GB RAM. The Integer Linear Program in our heuristic is solved by Matlab's Mixed-Integer Linear Programming solver and using the default configurations of the solver as specified in [51]. Observe that on average only few seconds were needed for our heuristic to make the migration decisions needed to avoid the PMs from overloading under the different parameters. The larger the number of ON PMs (N_{on}) and the sleeping PMs (N_{sleep}) that were considered as candidate PMs to host the migrated VMs by our heuristic, the higher the average time needed to make the necessary migrations, but also the lower the energy overhead of the made migrations as we have seen in Fig. 9.

By calculating the lower bound of the optimal solution we were also able to identify some of the cases where we know for sure that the migration decisions of our heuristic were optimal. Our heuristic's migration decisions were optimal any time a set of PMs were predicted to overload and the costs incurred by the decisions made by our heuristic to avoid the predicted overloads were exactly equal to the lower bound costs. We report in the third column of Table I how many times we know for sure that the migrations of our heuristic were optimal under the different parameter values where the reported numbers are percentages of the total times migrations needed to be performed to avoid some PMs from overloading during the entire 24-hour testing period. Observe that the percentages are very high and slightly increase as N_{on} and N_{sleep} increase as further PMs are considered as destination PMs to host the VMs selected for migration.

TABLE I
EVALUATION OF THE PROPOSED MIGRATION HEURISTIC UNDER DIFFERENT VALUES OF N_{on} AND N_{sleep} DURING THE ENTIRE TESTING PERIOD.

Heuristic Parameters	Mean Execution Time (second)	Percent of Times We Know that Migrations were Optimal
($N_{on} = 1$, $N_{sleep} = 1$)	1.8	88.4%
($N_{on} = 5$, $N_{sleep} = 5$)	2	96%
($N_{on} = 10$, $N_{sleep} = 10$)	3.4	97.1%

Number of Active PMs. Since the energy consumed by ON PMs constitutes a significant amount, we analyze in Fig. 10 the number of ON PMs when running our framework on the Google traces under each of the three studied migration heuristics. Recall that each migration heuristic makes different decisions to handle PM overloads, and these decisions affect the number of ON PMs, as new PMs may be switched ON to accommodate the migrated VMs. We also show the number of ON PMs when no overcommitment is applied. This represents the case when the exact amount of requested resources is allocated for each VM during its entire lifetime. By comparing these results, observe that after a couple of learning hours, our proposed prediction framework leads to smaller numbers of ON PMs when compared with the case of no overcommittment, and this is true regardless of the VM migration heuristic being used. Also, observe that our proposed prediction techniques, when coupled with our proposed VM migration heuristic, leads to the smallest number of ON PMs when compared with Largest First and Sandpipper heuristics,

[3]It is worth mentioning that the Knapsack-based constructed lower bound was preferred over the lower bound that can be obtained by solving the Linear Program (LP) relaxation of the optimization problem presented in Section V as our evaluations based on Google traces showed that our constructed lower bound was always tighter than the bound obtained by solving the LP relaxation. Furthermore, for the case when large number of PMs are predicted to overload, our constructed lower bound requires less amount of memory resources to calculate when compared to the LP relaxation lower bound.

resulting in greater energy savings. It is worth mentioning that during the first couple of hours, the number of ON PMs is the same regardless of whether resource overcommitment is employed and regardless of the migration technique being used, simply because prediction can't be beneficial at the early stage, as some time is needed to learn from past traces to be able to make good prediction about VMs' future utilizations.

Energy Savings. Fig. 11 shows the amount of energy (in Megawatt-Hour) that Google cluster saves every hour when adapting our integrated framework (the proposed prediction approach and the proposed migration heuristic) compared to no overcommitment. Observe that savings are not substantial at the beginning as the prediction module needs some time to learn the resource demands of the hosted VMs, but these savings quickly increase over time as the predictors start to observe larger traces and tune their parameters more accurately. It is also clear from Fig 11 that although our framework incurs migration energy overheads (due to both VM moving and PM switching energy overheads) that would not otherwise be present when no overcommitment is applied, the amount of energy saved due to the reduction of the number of ON PMs is much higher than the amount of energy incurred due to migration energy, leading, at the end, to greater energy savings.

Finally, the total energy that Google cluster consumes when adapting our integrated framework is compared to the case when the cluster is overcommited and uses one of the existing overload avoidance techniques, FUSD or Threshold-based, combined with each of the prior work migration heuristics, Largest First (LF) and Sandpiper (SP), to handle overloads. Fig. 12 shows the total cluster's energy consumption during the entire 24-hour duration normalized with respect to the no overcommitment case. Observe from Fig. 12 that our framework cuts the total consumed energy by 40% compared to the no overcommitment case. Observe also that the cluster's consumed energy when adapting our framework is significantly lower than any of the remaining overcommitment schemes. This is attributed to the fact that by predicting the resource demands of the hosted VMs accurately and by making efficient migration decisions, our integrated framework consolidates the workload using the least number of active PMs, while keeping redundant PMs in sleep state, which leads to significant lower energy consumption.

VIII. Conclusion and Future Work

We propose an integrated energy-efficient, prediction-based VM placement and migration framework for cloud resource allocation with overcommitment. We show that our proposed framework decreases the performance degradation caused by overloads while also reducing the number of PMs needed to be ON and the migration overheads, thereby making significant energy savings. All of our findings are supported by evaluations and comparative studies with existing techniques that were conducted on real traces from a Google cluster. For future work, we plan to conduct further comparative studies to evaluate our framework against other techniques and using further real traces.

References

[1] E. Gelenbe, "Energy packet networks: adaptive energy management for the cloud," in *Proceedings of the International Workshop on Cloud Computing Platforms*, 2012, p. 1.

[2] E. Gelenbe and C. Morfopoulou, "A framework for energy-aware routing in packet networks," *The Computer Journal*, vol. 54, no. 6, pp. 850–859, 2011.

[3] L. Barroso and U. Holzle, "The case for energy-proportional computing," *Computer Journal*, vol. 40, pp. 33–37, 2007.

[4] A. Berl, E. Gelenbe, M. Di Girolamo, G. Giuliani, H. De Meer, M. Dang, and K. Pentikousis, "Energy-efficient cloud computing," *The computer journal*, vol. 53, no. 7, pp. 1045–1051, 2010.

[5] C. Reiss, J. Wilkes, and J. Hellerstein, "Google cluster-usage traces: format+ schema," *Google Inc., White Paper*, 2011.

[6] M. Dabbagh, B. Hamdaoui, M. Guizani, and A. Rayes, "Towards energy-efficient cloud computing: Prediction, consolidation, and overcommitment," *IEEE Network Magazine*, 2015.

[7] L. Tomas and J. Tordsson, "An autonomic approach to risk-aware data center overbooking," *IEEE Transactions on Cloud Computing*, vol. 2, no. 3, pp. 292–305, 2014.

[8] Haikun L., Hai J., Xiaofei L., Chen Y., and Cheng-Zhong X., "Live virtual machine migration via asynchronous replication and state synchronization," *IEEE Transactions on Parallel and Distributed Systems*, vol. 22, no. 12, pp. 1986–1999, 2011.

[9] E. Gelenbe and R. Lent, "Energy–QoS trade-offs in mobile service selection," *Future Internet*, vol. 5, no. 2, pp. 128–139, 2013.

[10] L. Wang and E. Gelenbe, "Adaptive dispatching of tasks in the cloud," *IEEE Tansactions on Cloud Computing*, 2015.

[11] E. Gelenbe, "Adaptive management of energy packets," in *IEEE Computer Software and Applications Conference Workshops*, 2014, pp. 1–6.

[12] E. Gelenbe, "Energy packet networks: smart electricity storage to meet surges in demand," in *International Conference on Simulation Tools and Techniques*, 2012, pp. 1–7.

[13] E. Gelenbe and C. Morfopoulou, "Power savings in packet networks via optimised routing," *Mobile Networks and Applications*, vol. 17, no. 1, pp. 152–159, 2012.

[14] G. Sakellari, C. Morfopoulou, and E. Gelenbe, "Investigating the tradeoffs between power consumption and quality of service in a backbone network," *Future Internet*, vol. 5, no. 2, pp. 268–281, 2013.

[15] M. Dabbagh, B. Hamdaoui, M. Guizani, and A. Rayes, "Energy-efficient resource allocation and provisioning framework for cloud data centers," *IEEE Transactions on Network and Service Management*, 2015.

[16] D. Gmach, J. Rolia, L. Cherkasova, and A. Kemper, "Resource pool management: Reactive versus proactive or let's be friends," *Computer Networks*, vol. 53, no. 17, pp. 2905–2922, 2009.

[17] X. Wang and Y. Wang, "Coordinating power control and performance management for virtualized server clusters," *IEEE Transactions on Parallel Distributed Systems*, vol. 22, no. 2, pp. 245–259, 2011.

[18] S. Kumar, V. Talwar, V. Kumar, P. Ranganathan, and K. Schwan, "vmanage: loosely coupled platform and virtualization management in data centers," in *Int'l Conf. on Autonomic computing*, 2009.

[19] A. Gulati, A. Holler, M. Ji, G. Shanmuganathan, C. Waldspurger, and X. Zhu, "VMware distributed resource management: Design, implementation, and lessons learned," *VMware Technical Journal*, vol. 1, no. 1, pp. 45–64, 2012.

[20] R. Ghosh and V. Naik, "Biting off safely more than you can chew: Predictive analytics for resource over-commit in iaas cloud," in *IEEE International Conference on Cloud Computing (CLOUD)*, 2012.

[21] D. Gmach, J. Rolia, L. Cherkasova, G. Belrose, T. Turicchi, and A. Kemper, "An integrated approach to resource pool management: Policies, efficiency and quality metrics," in *International Conference on Dependable Systems and Networks*, 2008.

[22] G. Dhiman, G. Marchetti, and T. Rosing, "vGreen: a system for energy efficient computing in virtualized environments," in *ACM international symposium on Low power electronics and design*, 2009.

[23] 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*, 2012.

[24] A. Beloglazov and R. Buyya, "Adaptive threshold-based approach for energy-efficient consolidation of virtual machines in cloud data centers,"

in *Proceedings of ACM International Workshop on Middleware for Grids, Clouds and e-Science*, 2010, p. 4.
[25] M. Ramani and M. Bohara, "Energy aware load balancing in cloud computing using virtual machines," *Journal of Engineering Computers & Applied Sciences*, vol. 4, no. 1, pp. 1–5, 2015.
[26] N. Bobroff, A. Kochut, and K. Beaty, "Dynamic placement of virtual machines for managing sla violations," in *IEEE International Symposium on Integrated Network Management*, 2007.
[27] R. Chiang, J. Hwang, H. Huang, and T. Wood, "Matrix: achieving predictable virtual machine performance in the clouds," in *the International Conference on Autonomic Computing (ICAC)*, 2014.
[28] L. Chen and H. Shen, "Consolidating complementary VMs with spatial/temporal-awareness in cloud datacenters," in *Proceedings of IEEE INFOCOM*, 2014, pp. 1033–1041.
[29] B. Farhang-Boroujeny, *Adaptive filters: theory and applications*, John Wiley & Sons, 2013.
[30] G. Lindgren, H. Rootzén, and M. Sandsten, *Stationary stochastic processes for scientists and engineers*, CRC press, 2013.
[31] Z. Xiao, W. Song, and Q. Chen, "Dynamic resource allocation using virtual machines for cloud computing environment," *IEEE Transactions on Parallel and Distributed Systems*, vol. 24, no. 6, pp. 1107–1117, 2013.
[32] C. Clark, K. Fraser, S. Hand, J. Hansen, E. Jul, C. Limpach, I. Pratt, and A. Warfield, "Live migration of virtual machines," in *Symposium on Networked Systems Design & Implementation*, 2005.
[33] O. Abdul-Rahman, M. Munetomo, and K. Akama, "Live migration-based resource managers for virtualized environments: a survey," in *CLOUD COMPUTING 2010, The First International Conference on Cloud Computing, GRIDs, and Virtualization*, 2010, pp. 32–40.
[34] A. Celesti, F. Tusa, M. Villari, and A. Puliafito, "Improving virtual machine migration in federated cloud environments," in *International Conference on Evolving Internet*, 2010.
[35] M. Andreolini, S. Casolari, M. Colajanni, and M. Messori, "Dynamic load management of virtual machines in cloud architectures," in *Cloud Computing*, pp. 201–214. Springer, 2010.
[36] N. Bobroff, A. Kochut, and K. Beaty, "Dynamic placement of virtual machines for managing sla violations," in *International Symposium on Integrated Network Management*, 2007.
[37] X. Zhang, Z. Shae, S. Zheng, and H. Jamjoom, "Virtual machine migration in an over-committed cloud," in *Network Operations and Management Symposium (NOMS)*, 2012.
[38] V. Shrivastava, P. Zerfos, K. Lee, H. Jamjoom, Y. Liu, and S. Banerjee, "Application-aware virtual machine migration in data centers," in *Proceedings of IEEE INFOCOM*, 2011.
[39] X. Meng, V. Pappas, and L. Zhang, "Improving the scalability of data center networks with traffic-aware virtual machine placement," in *Proceedings of IEEE INFOCOM*, 2010.
[40] N. Jain, I. Menache, J. Naor, and F. Shepherd, "Topology-aware vm migration in bandwidth oversubscribed datacenter networks," in *Automata, Languages, and Programming*, pp. 586–597. Springer, 2012.
[41] T. Wood, P. Shenoy, A. Venkataramani, and M. Yousif, "Sandpiper: Black-box and gray-box resource management for virtual machines," *Computer Networks*, vol. 53, no. 17, pp. 2923–2938, 2009.
[42] M. Dabbagh, B. Hamdaoui, M. Guizani, and A. Rayes, "Energy-efficient cloud resource management," in *INFOCOM Workshop on Moblie Cloud Computing*, 2014.
[43] R. Panigrahy, K. Talwar, L. Uyeda, and U. Wieder, "Heuristics for vector bin packing," *Microsoft Research Report*, 2011.
[44] H. Liu, C. Xu, H. Jin, J. Gong, and X. Liao, "Performance and energy modeling for live migration of virtual machines," in *international symposium on High performance distributed computing*, 2011.
[45] E. Gelenbe, R. Lent, and M. Douratsos, "Choosing a local or remote cloud," in *IEEE Symposium on Network Cloud Computing and Applications (NCCA)*, 2012, pp. 25–30.
[46] E. Gelenbe and R. Lent, "Trade-offs between energy and quality of service," in *the IFIP Conference on Sustainable Internet and ICT for Sustainability*, 2012, pp. 1–5.
[47] M. Dabbagh, B. Hamdaoui, M. Guizani, and A. Rayes, "Release-time aware vm placement," in *Globecom Workshops (GC Wkshps), 2014*. IEEE, 2014, pp. 122–126.
[48] M. Dabbagh, B. Hamdaoui, M. Guizani, and A. Rayes, "Online assignment and placement of cloud task requests with heterogeneous requirements," in *IEEE International Global Communication Conference*. IEEE, 2015.
[49] M. Dabbagh, B. Hamdaoui, M. Guizani, and A. Rayes, "Exploiting task elasticity and price heterogeneity for maximizing cloud computing profits," *IEEE Transactions on Emerging Topics in Computing*, 2015.
[50] A. Fréville, "The multidimensional 0–1 knapsack problem: An overview," *European Journal of Operational Research*, vol. 155, no. 1, pp. 1–21, 2004.
[51] Matlab Mixed-Integer Linear Programming Solver, *http://www.mathworks.com/help/optim/ug/intlinprog.html*.

Mehiar Dabbagh received his B.S. degree in Telecommunication Engineering from the University of Aleppo, Syria, in 2010 and the M.S. degree in ECE from the American University of Beirut (AUB), Lebanon, in 2012. Currently, he is pursuing the Ph.D. degree in EECS at Oregon State University, USA. During his PhD studies, he interned with Cisco and with HP where he worked on developing cloud-related technologies. His research interests include: Cloud Computing, Distributed Systems, Energy Efficiency, Networking Security and Data Mining.

Bechir Hamdaoui (S'02-M'05-SM'12) is presently an Associate Professor in the School of EECS at Oregon State University. He received the Diploma of Graduate Engineer (1997) from the National School of Engineers at Tunis, Tunisia. He also received M.S. degrees in both ECE (2002) and CS (2004), and the Ph.D. degree in Computer Engineering (2005) all from the University of Wisconsin-Madison. His current research focus is on distributed optimization, parallel computing, cognitive networking, cloud computing, and Internet of Things. He has won the NSF CAREER Award (2009), and is presently an AE for IEEE Transactions on Wireless Communications (2013-present), and Wireless Communications and Computing Journal (2009-present). He also served as an AE for IEEE Transactions on Vehicular Technology (2009-2014) and for Journal of Computer Systems, Networks, and Communications (2007-2009). He served as the chair for the 2011 ACM MobiCom's SRC program, and as the program chair/co-chair of several IEEE symposia and workshops (including GC 2016, ICC 2014, IWCMC 2009-2016, CTS 2012, PERCOM 2009). He also served on technical program committees of many IEEE/ACM conferences, including INFOCOM, ICC, GLOBECOM, and others. He has recently been selected as a Distinguished Lecturer for the IEEE Communication Society for 2016 and 2017. He is a Senior Member of IEEE, IEEE Computer Society, IEEE Communications Society, and IEEE Vehicular Technology Society.

Mohsen Guizani (S'85-M'89-SM'99-F'09) is currently a Professor at the Computer Science & Engineering Department in Qatar University. Qatar. He also served in academic positions at the University of Missouri-Kansas City, University of Colorado-Boulder, Syracuse University and Kuwait University. He received his B.S. (with distinction) and M.S. degrees in EE; M.S. and Ph.D. degrees in CS in 1984, 1986, 1987, and 1990, respectively, all from Syracuse University, Syracuse, New York. His research interests include Wireless Communications, Computer Networks, Cloud Computing, Cyber Security and Smart Grid. He currently serves on the editorial boards of several international technical journals and the Founder and EiC of "Wireless Communications and Mobile Computing" Journal published by John Wiley. He is the author of nine books and more than 400 publications in refereed journals and conferences (with an h-index=30 according to Google Scholar). He received two best research awards. Dr. Guizani is a Fellow of IEEE, member of IEEE Communication Society, and Senior Member of ACM.

Ammar Rayes Ph.D., is a Distinguished Engineer at Cisco Systems and the Founding President of The International Society of Service Innovation Professionals, www.issip.org. He is currently chairing Cisco Services Research Program. His research areas include: Smart Services, Internet of Everything (IoE), Machine-to-Machine, Smart Analytics and IP strategy. He has authored / co-authored over a hundred papers and patents on advances in communications-related technologies, including a book on Network Modeling and Simulation and another one on ATM switching and network design. He is an Editor-in-Chief for "Advances of Internet of Things" Journal and served as an Associate Editor of ACM "Transactions on Internet Technology" and on the Journal of Wireless Communications and Mobile Computing. He received his BS and MS Degrees in EE from the University of Illinois at Urbana and his Doctor of Science degree in EE from Washington University in St. Louis, Missouri, where he received the Outstanding Graduate Student Award in Telecommunications.