Energy-Efficient Cloud Resource Management

Mehiar Dabbagh, Bechir Hamdaoui, Mohsen Guizani† and Ammar Rayes‡
Oregon State University, Corvallis, OR 97331, dabbaghm,hamdaoub@onid.orst.edu
† Qatar University, mguizani@ieee.org
‡ Cisco Systems, San Jose, CA 95134, ‡ rayes@cisco.com

***Abstract*—We propose a resource management framework that reduces energy consumption in cloud data centers. The proposed framework predicts the number of virtual machine requests along with their amounts of CPU and memory resources, provides accurate estimations of the number of needed physical machines, and reduces energy consumption by putting to sleep unneeded physical machines. Our framework is based on real Google traces collected over a 29-day period from a Google cluster containing over 12,500 physical machines. Using this Google data, we show that our proposed framework makes substantial energy savings.**

***Index Terms*—Cloud computing, energy efficiency, cloud data centers, cloud data clustering, cloud load prediction.**

I. Introduction

Data centers consumed about 1.5% of the total generated electricity in the U.S. in 2006, an amount that is equivalent to the annual energy consumption of 5.8 million households [1]. This consumption is expected to increase even further as data centers are anticipated to grow in both size and numbers. For example, a recent study by Cisco predicts that cloud traffic will grow 12-fold by 2015 [2]. It is therefore important to develop efficient techniques to reduce data center energy consumption.

Large data centers such as cloud centers consist typically of tens of thousands of servers, often referred to as physical machines (PMs) and grouped into clusters. Upon receiving a client request, the cluster scheduler creates a virtual machine (VM) for the requested resources and assigns it to a PM. According to a Google study [3], idle PMs consume around 50% of their peak power. Therefore, it is very important to switch PMs to the sleep mode whenever they are not in use in order to save energy. In this paper, we propose an integrated resource management framework that predicts VM requests to make energy-efficient resource management decisions.

Approaches that predict workloads to allocate resources efficiently have already been proposed in the context of distributed systems, such as Grid systems. Hidden Markov models [4], polynomial fitting [5], and hybrid models [6] are workload prediction methods that have been used in Grid systems. As for cloud centers, machine learning techniques, such as Neural Networks, have been proposed [7, 8] to predict the number of requests that could arrive at cloud-hosted web servers. The problem being addressed in this paper is different from these previous works, as not only do we predict the number of VM requests to arrive at a cloud cluster, but also the requested amount of each type of resources associated with each VM request. Specifically, our proposed framework i) combines machine learning clustering and stochastic theory to predict future VM request loads, ii) provides accurate estimations of the number of PMs to be needed in the future, and iii) makes energy-efficient resource management decisions to reduce energy consumption in cloud data centers. Our framework is evaluated through real Google traces [9] collected over a 29-day period from a Google cluster containing over 12,500 PMs.

This work was supported in part by Cisco (CG-573228) and National Science Foundation (CAREER award CNS-0846044)

The rest of the paper is organized as follows. In Section II, we overview our proposed framework. In Section III, we present the Google data traces and clustering. In Section IV, we present our prediction approach. In Section V, we evaluate our proposed framework. Finally, we conclude the paper in Section VI.

II. An Overview of Proposed Framework

Our framework has three major components: data clustering, workload prediction, and power management. In this section, we briefly describe these components so as the reader will have a global picture of the entire framework before delving into the details. Detailed descriptions are provided in later sections. Throughout this section, we refer to Fig. 1 for illustration.

A. Data Clustering

Our prediction approach relies on observing and monitoring workload variations during a past time period, referred to as the *observation window*, in order to predict the workload coming in a certain future time period, referred to as the *prediction window*. A VM request typically consists of multiple different cloud resources (e.g., CPU, memory, bandwidth, etc.) with different amounts. This multi-resource nature of these VM requests poses unique challenges when it comes to developing prediction techniques. Also, different cloud clients may request different amounts of the same resources. Therefore, it is both unpractical and too difficult to predict the demand of each type of resource separately (though ideally this is what is needed to be able to make optimal power management decisions). To overcome these challenges, we instead first divide/group VM requests into several categories, and then develop prediction techniques for each of these categories. This is known as clustering.

1) k-Means clustering: Our first step is then to create a set of clusters to contain all types of VM requests; i.e., each VM request is mapped into one and only one cluster, and all requests belonging to the same cluster possess similar characteristics in terms of requested resources. Our clusters are created based on real Google data traces [9]. Since these Google traces consider only two types of resources, CPU and memory, we also consider only these two types in our framework. In order to divide requests into multiple categories, we represent them in the $\mathbb{R}^2$ space, where each point is a request and the two dimensional coordinates of the point are the amounts of CPU and memory resources associated with the request. Clustering these data points into a number of clusters is done using k-Means [10].

As shown in Fig. 1, the k-Means algorithm takes as an input the Google traces and the number of clusters, k, and outputs k

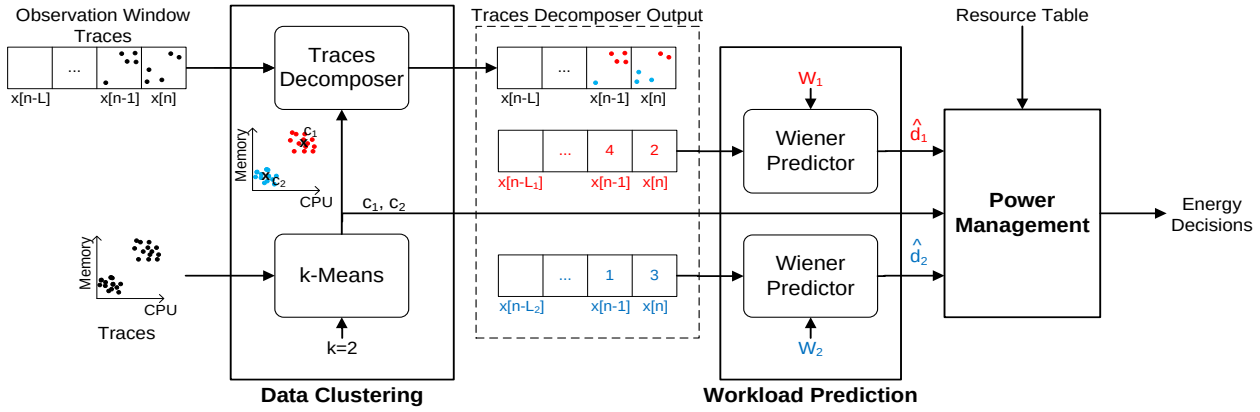

Fig. 1. Flow chart of the proposed framework

clusters, each specified by its center point (in the example given in the figure, $k = 2$ and the two clusters are shown in Red and Blue). Note that the parameter k needs to be chosen *a priori* and given as an input to the clustering algorithm. In Section III-B, we show how such a parameter is chosen.

2) Traces Decomposer: Once the k clusters and their center points are determined, they are given as an input to the Traces Decomposer module (shown in Fig. 1), which is responsible for mapping each request received during the observation window into one cluster. The observation window is split into $L+1$ time slots, $n, n-1, \ldots, n-L$, as follows. Suppose a prediction needs to be made at time t. In this case, slot n corresponds to time interval $[t-10, t]$ (in seconds); slot $n-1$ corresponds to time interval $[t-20, t-10]$, slot $n-i$ corresponds to time interval $[t-10(i+1), t-10i]$, and so on. The Traces Decomposer tracks the number $x[n-i]$ of received requests in time slot $[t-10(i+1), t-10i]$ of the observation window for all $i = 0, 1, \ldots, L$, and maps each request within the slots into one cluster.

B. Workload Prediction

We use stochastic Wiener filter prediction to estimate the workload of each category/cluster. The Stochastic Predictor, as shown in Fig. 1, is made of k Wiener filters. Each filter takes as an input the number of received requests for a certain category during the observation window, and uses it to predict the number of requests of that same category to arrive in the prediction window. This makes the problem easier to solve as there is an infinite number of possible combinations of the amounts of CPU and memory that a client may request.

The prediction for each category is a weighted linear combination of the number of requests of that category observed during the observation window. The parameters that need to be determined for each filter branch are: the length of the observation window, the length of the prediction window, and the weight vector. These parameters are determined in Section IV.

C. Power Management

The predictions of all categories along with their center points are all next passed to the Power Management module, which uses this information to decide which PMs need to go to sleep and which ones need to be kept ON. This unit keeps track of current utilizations and states (ON or sleep) of all PMs in the cloud cluster which are stored in a table called Resource Table. It uses a modified Best Fit Decreasing (BFD) heuristic [11] to fit the predicted VM requests in PMs in order to determine the number of PMs to be needed during the next prediction window period.

The original BFD algorithm [11] tries to pack VM requests in the fullest PM with enough space. In order to do that, it sorts PMs from the fullest to the least full and iterates over the ordered list of PMs trying to pack the VM request within the first PM that has enough space. The original algorithm can not be applied directly to our framework due to the following limitations. First, it was introduced for homogeneous bins where all bins (i.e., PMs) are assumed to have the same capacity. In cloud centers, different PMs may have different capacities, as will be seen in Section III. Second, it only considers one dimension (i.e., one resource only), whereas our cloud centers consider two dimensions, CPU and memory. We overcome the first limitation by considering the PM's utilization as the metric for measuring how full a PM is, and overcome the second limitation by mapping the two dimensions into a single-dimension metric that combines both dimensions. The single-dimension metric considered in our heuristic is the product of the two dimensions. Furthermore, our proposed heuristic takes energy efficiency into account by sorting the PMs according to the following criteria (in ascending order):

(i) PMs that are ON
(ii) PMs that have higher utilizations. The utilization metric is defined as the product of the CPU utilization and the memory utilization of the PM.
(iii) PMs that have higher capacities. Similarly, the capacity metric of a PM is defined as the product of its memory and CPU capacities.

The intuition behind our sorting criteria is as follows: we want to make use of the available ON PMs that already have some scheduled VMs; so ON PMs are ranked first. We then use the utilization metric as the next sorting criterion, since PMs are more energy efficient when they have higher utilization [3]. So, it is better to increase their utilization when scheduling new VMs. Finally, PMs are sorted based on their capacities, as one can fit more VMs in PMs that have large capacities. After placing all predicted VM requests in PMs, we either end up with redundant ON PMs that need to be turned to the sleep mode, or more PMs will be needed in the future and therefore are turned back ON from the sleep mode to cover the predicted workload. This guarantees that only PMs needed for the predicted workload are

to be kept ON and the rest are put to sleep to save energy.

III. Data Clustering

In this section, we first begin by presenting the Google data traces that we used in this work to train and test our developed energy-aware resource provisioning models, and then present our workload clustering and classification findings.

A. Google Traces

We conduct our experiments on real Google data [9] that was released in November 2011 and consists of a 29-day traces collected from a cluster that contains more than 12,500 PMs. To the best of our knowledge, this is the first work that considers developing a workload prediction approach based on this data.

The Google cluster supports different PM configurations. The characteristics and number of PMs of each configuration are described in Table I. Note that there are three different types of architecture, labeled by Google as A, B and C (their actual types are not given for privacy reasons). Note also that PMs from the same architecture may have different CPU and memory capacities. Only normalized capacities (w.r.t. the maximum capacity) are provided, also for privacy reasons; thus the reported capacities are all less than or equal to one.

TABLE I
CONFIGURATIONS OF THE PMS WITHIN THE GOOGLE CLUSTER

Number of PMs	PM Configurations		
	Architecture	CPU	Memory
6732	A	0.50	0.50
3863	A	0.50	0.25
1001	A	0.50	0.75
795	C	1.00	1.00
126	B	0.25	0.25
52	A	0.50	0.12
5	A	0.50	0.03
5	A	0.50	0.97
3	C	1.00	0.50
1	A	0.50	0.06

Our experiments utilize the data provided in the task event table, where each VM request is called a task and each VM submission/termination request is referred to as an event. A detailed description of the data is provided in [12], and in the following, we only describe the features used by our framework:

- Timestamp: time at which the event happened.
- VM ID: a unique identifier for each VM. Kept anonymous and replaced by its hash value.
- Client ID: a unique identifier for each cloud client. Kept anonymous and replaced by its hash value.
- Event type: specifies whether the event is a submission or a release request. It is worth noting that clients may submit or release a VM request whenever they desire.
- Requested CPU: amount of requested CPU.
- Requested memory: amount of requested memory.

Since the size of the Google data is huge (compressed size is about 39GB), we use two (different) chunks of the traces to tune our framework parameters. We refer to these chunks as the *training data set* and the *validation data set*, and are of length 24 and 16 hours, respectively. A different *testing data set*, also taken from the Google traces, with a duration of 29 hours is used later to estimate the accuracy and measure energy savings that our framework achieves.

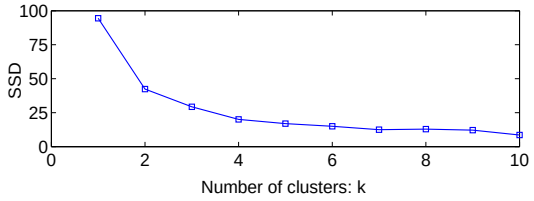

Fig. 2. The elbow criteria for selecting k.

B. Workload Clustering

We use k-Means, a well-known unsupervised learning algorithm, to cluster VM requests into k categories [10]. k-Means assigns N data points to k different clusters, where k is a parameter that needs to be specified *a priori*. The number of clusters, k, is one of the important parameters that needs to be tuned when using k-Means. For this, a heuristic approach is implemented, in which the Sum of Squared Distances (SSD) is plotted as a function of k. SSD represents the error when each point in the data set is represented by its corresponding cluster center, and is mathematically equal to $\sum_{i=1}^{k}\sum_{r\in\mathcal{C}_i}\mathrm{d}(r,c_i)^2$ where $\mathcal{C}_i$ denotes cluster i; i.e., set of all points belonging to the i^{th} cluster, c_i denotes cluster i's center point, and $\mathrm{d}(r,c_i)$ is the Euclidean distance between r and c_i.

Fig. 2 shows SSD as a function of k plotted based on the Google traces. Note that as k increases, SSD decrease monotonically, and hence so does the error. Recall that while increasing k reduces the error, it also increases the overhead incurred by the prediction technique (to be presented in next sections), since a predictor branch needs to be built for each cluster/category. For this, the heuristic searches then for the "elbow" or "knee" point of the plot, which is basically the point that balances between these two conflicting objectives: reducing errors and maintaining low overhead. As can be seen from Fig. 2, the value 4 for k strikes a good balance between accuracy and overhead. Hence, in what follows we use $k = 4$.

Fig. 3 shows the resulting clusters for $k = 4$, where each category is marked by a different color/shape and the centers of these clusters c_1, c_2, c_3 and c_4 are marked by 'x'. Category 1 represents VM requests with small amounts of CPU and small amounts of memory; Category 2 represents VM requests with medium amounts of CPU and small amounts of memory; Category 3 represents VM requests with large amounts of memory (and any amounts of requested CPU). Category 4 represents VM requests with large amounts of CPU (and any amounts of requested memory). Observe from the obtained clusters that requests with smaller amounts of CPU and memory are denser than those with large amounts.

IV. Workload Prediction

In this section, we determine and estimate the parameters of the proposed prediction approach.

A. Length of the Prediction Window

An important parameter that needs to be estimated for the stochastic predictor is the length of the prediction window, T_p. This is the time period for which the workload needs to be predicted to decide whether PMs need to be switched to the sleep mode. When a PM is switched to the sleep mode, then

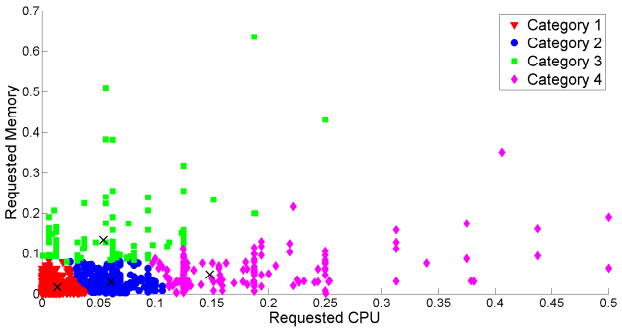

Fig. 3. The resulting four clusters/categories for Google traces.

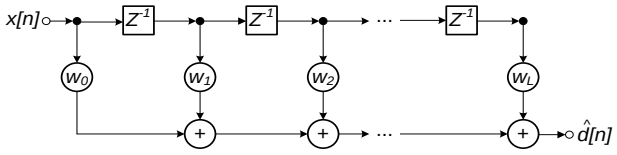

Fig. 4. The general structure of Wiener filter.

its consumed energy while in the sleep mode, E_{sleep}, is equal to $E_o + P_{sleep}(T_p - T_o)$, where P_{sleep} is the consumed power when in the sleep mode; E_o is the transition energy, equaling the energy needed to switch the machine to the sleep mode plus the energy needed to wake up the machine later; and T_o is the transitional switching time. Now let T_{be} be the break-even time, the time during which keeping the PM ON and idle consumes an amount of energy that is equal to E_{sleep}. We then have $P_{idle}T_{be} = E_o + P_{sleep}(T_{be} - T_o)$, where P_{idle} is the power the PM consumes while being ON and idle. Putting a PM to sleep saves energy only when the PM stays idle for a period longer than T_{be}; i.e., when $T_p \geq T_{be}$.

In this work, we rely on the energy measurements conducted in [13] to estimate T_{be}. These measurements, shown in Table II, yield $T_{be} = 47$ seconds. It is worth noting that these energy measurements were based on a certain type of commercial PMs, and hence, these numbers might change slightly depending on the PM type. In our framework, we chose a conservative value of $T_p = 60$ seconds so as to accommodate for all types. But as mentioned before, Google does not provide information about the types of their PMs; so this number can still be off.

TABLE II
ENERGY MEASUREMENTS NEEDED TO CALCULATE T_{be}

Parameter	Value	Unit
P_{idle}	300.81	Watt
P_{sleep}	107	Watt
$E_{on \to sleep}$	5510	Joule
$E_{sleep \to on}$	4260	Joule
T_o	6	Seconds

B. Weights of the Stochastic Predictor

Suppose that the prediction needs to be made at time t. The general structure of the Wiener predictor for each of the four categories can be represented as shown in Fig. 4, where:

- $x[n-i]$: is the number of requests received in the period between $t - 10(i+1)$ and $t - 10i$ seconds.
- $d[n]$: the desired output of the category predictor. This represents the actual number of requests for the considered category in the coming prediction window.
- $\hat{d}[n]$: is the predicted number of requests for the considered category in the coming prediction window.
- L: is the number of taps that the predictor relies on in making predictions.
- w_i: is the i^{th} tap's weight.

Wiener filter predicts the future requests, assuming $x[n]$ is a wide-sense stationary process. The predicted number of requests, $\hat{d}[n]$, is a weighted average of the previous observed requests; i.e., $\hat{d}[n] = \sum_{i=0}^{L} w_i x[n-i]$. The prediction error, $e[n]$, can be calculated as the difference between the actual and predicted number of requests; i.e., $e[n] = d[n] - \hat{d}[n] = d[n] - \sum_{i=0}^{L} w_i x[n-i]$. The objective is to find the weights that minimize the Mean Squared Error (MSE) of the training data, where $MSE = E[\,e^2[n]\,]$ represents the second moment of error. The reason for choosing it as an objective is because it minimizes both the average and variance of the error while putting more weight on the average ($E[e^2[n]] = (\,E[\,e[n]\,]\,)^2 + var(\,e[n]\,)$). Differentiating MSE with respect to w_k and setting this derivative to zero yields, after some algebraic simplifications, $E\big[d[n]x[n-k]\big] - \sum_{i=0}^{L} w_i E\big[x[n-k]x[n-i]\big] = 0$. By letting

$$R_{xx}(i-k) = E\big[x[n-k]x[n-i]\big] \quad (1)$$

$$R_{dx}(k) = E\big[d[n]x[n-k]\big] \quad (2)$$

it follows that $R_{dx}(k) = \sum_{i=0}^{L} w_i R_{xx}(i-k)$.

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

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

$$W = \begin{bmatrix} w_0 & w_1 & \dots & w_L \end{bmatrix}^T$$

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

Given R_{xx} and R_{dx}, the weights can then be calculated as $W = R_{xx}^{-1} R_{dx}$. We rely on the training data set to calculate R_{xx} and R_{dx} for each category. To estimate these parameters for a certain category, we divide the training data into N slots where the duration of each slot is 10 seconds. We first calculate the number of requests of the considered category that are received in each slot. Then, we calculate the elements of R_{xx} using the unbiased correlation estimation as:

$$R_{xx}(m) = \frac{1}{N-m} \sum_{j=0}^{N-m-1} x[j+m]x[j]$$

The elements of R_{dx} can also be estimated using the correlation coefficients. Since $d[n]$ represents the number of requests in the coming prediction window which has a duration of 60 seconds, we can write $d[n] = \sum_{i=1}^{6} x[n+i]$. Plugging the expression of $d[n]$ in Eq. (2) yields the correlations that can be estimated from the training data. An estimation of the weight vector follows then for each category predictor provided R_{dx} and R_{xx} are known. These weights lead, in turn, to the lowest MSE for the training data.

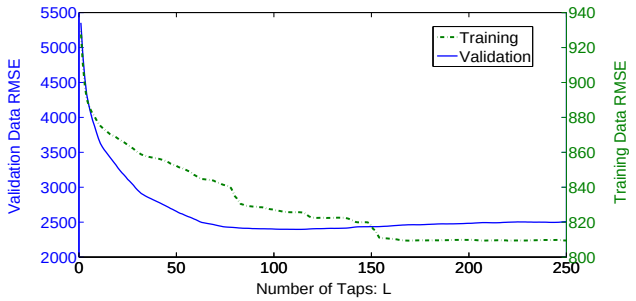

Fig. 5. RMSE for 3rd category predictor

C. Length of the Observation Window

The last parameter that needs to be tuned for each category predictor is the length of the observation window. As mentioned before, the observation window is divided into L slots or (also called) taps, each tap/slot is of length 10 seconds. This Wiener filter is referred to as an L-tapped filter. To determine L for a category predictor, we first need to find the optimal weight vectors of the Wiener predictor under different values of L on the training data, and then test the performance of these weight vectors on the unseen validation data.

Fig. 5 shows the Root Mean Square Error ($RMSE$) of the 3rd category predictor for both training and validation data sets for different values of L. Observe that the training data $RMSE$ decreases monotonically as L increases. To understand this, consider two Wiener filters: one with L taps and the other with $L + R$ taps. Recall that we need to find optimal weights that when multiplied by these taps leads to the minimum MSE of the training data. As a result, the model with $L + R$ taps can achieve the same accuracy on the training data set as the model with L taps by setting the weights of all the additional R taps to 0. Thus, the model with a given number of taps will, in the worst-case scenario, achieve the same accuracy as any model with lower numbers of taps. In general, models with larger numbers of taps can still find some correlations specific to the training data that lead to a better accuracy. Consequently, the training error continues to decrease as the number of taps increases.

However, by observing the behavior of the validation data, note that $RMSE$ decreases first until it reaches a point beyond which the error can no longer be reduced even if L is increased further. Also, observe that if we continue to increase L, the validation data $RMSE$ starts to increase. This behavior is expected and is known as the overfitting phenomenon. After increasing the number of taps beyond a certain limit, the model tries to find correlations between the different requests over time. These correlations are specific to the training data. We say then that increasing the number of taps beyond a certain point results in increasing the complexity of the model and starts finding correlations that do not exist in the general traces but are specific to the training data. Based on our experiments, we chose $L = 80$, as it achieves the best accuracy on the unseen validation data, meaning that the observation window of the 3rd category predictor relies on the traces in the previous 80 taps collected in the previous $80 \times 10 = 800$ seconds.

Using a similar approach, the optimal numbers of taps of the other three categories are determined to be 20 (category 1), 80 (category 2), and 34 (category 4). Graphs for these three categories are omitted due to space limitation.

TABLE III
RMSE OF DIFFERENT PREDICTIVE APPROACHES.

	Wiener Predictor	Min Predictor	Average Predictor	Last Minute Predictor	Max Predictor
RMSE	2497	2547	2575	3117	5226

D. Prediction Accuracy

We assess the accuracy of the proposed predictors and compare it against those of the following four basic techniques:

- Last minute predictor: returns the same number of requests observed during the previous minute.
- Average Predictor: observes the number of requests received in each minute during the last five minutes, and returns the average over these five observations.
- Min Predictor: observes the number of requests received in each minute during the last five minutes, and returns the minimum of these five observations.
- Max Predictor: observes the number of requests received in each minute during the last five minutes, and returns the maximum of these five observations.

Table III shows RMSE of each prediction approach. For each approach, $RMSE = \sum_{i=1}^{4} RMSE_i$ where $RMSE_i$ is the approach's RMSE corresponding to the ith category. These evaluations are conducted on a testing data of Google traces that includes 2.5 milion VM requests received during a 29-hour period. The testing data set is different from the training and validation data sets used for tuning the parameters, thus providing a fair comparison by showing the performance of our predictor over new data that it did not see before. The table shows that the proposed Wiener predictor yields RMSE that is lower than those of all the other approaches. We want to mention that these basic prediction techniques have also been tested for different time durations (not only 5 minutes, but also 10, 15 minutes, etc.). For all of these cases, the basic approaches achieve a performance worse than that of the Wiener predictor. Due to space limitation, results reported in Table III consider the number of requests received only during the last 5-minute duration.

E. Safety Margin

Our stochastic predictors can still be a little off, leading to an under- or over-estimation of the number of requests. Under-estimating the number of future requests results in extra delays when allocating cloud resources to clients due to the need for waking up machines upon arrival of any unpredicted request(s). In order to reduce the occurrences of such cases, we add a safety margin to accommodate for such variations. The cost of this safety margin is that some PMs will need to be kept idle even though they may or may not be needed. We use a dynamic approach for selecting the appropriate safety margin value. The value depends on the accuracy of our predictors, and increases when the predictions deviate much from the actual number of requests and decreases otherwise. Since these deviations may vary over time, we calculate an exponentially weighted moving average (EWMA) of the deviations while giving higher weights

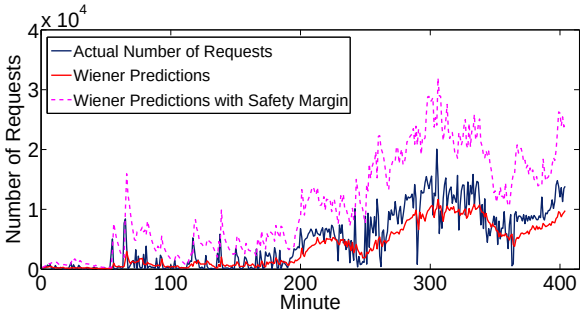

Fig. 6. Wiener prediction with and without safety margin for category 3.

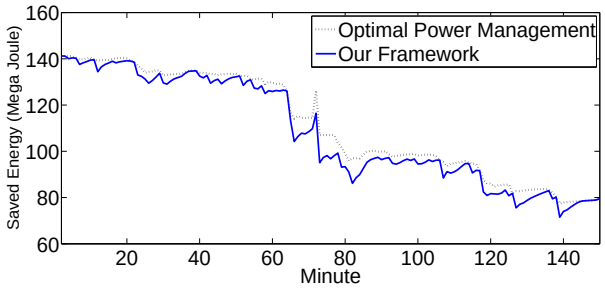

Fig. 7. Energy savings.

to the most recent ones. Initially, Dev is set to zero; i.e., $Dev[0] = 0$, and for later time n, $Dev[n]$ is updated as

$$Dev[n] = (1-\alpha)Dev[n-1] + \alpha \left| d[n-1] - \hat{d}[n-1] \right|$$

where $0 < \alpha < 1$ is the typical EWMA weight factor used to tune between the weight given to most recent deviations over that given to the previous ones. Dev is updated before the next prediction is made by observing the deviation between the predicted and actual number of requests in the previous minute. One advantage of EWMA is that the moving average is calculated without needing to store all observed deviations.

We add the weighted average Dev to our predictions in the next minute after we multiply it by a certain parameter β. The predicted number of requests with safety margin $\bar{d}[n]$ can then be calculated as $\bar{d}[n] = \hat{d}[n] + \beta Dev[n]$.

Fig. 6 shows the predicted and actual numbers of requests with and without safety margin for the third category (Graphs for the remaining categories are omitted due to space limitation). We set $\alpha = 0.25$ and $\beta = 4$; these values are selected experimentally by picking the values that provide the best predictions. Note from these figures that the prediction with safety margin forms an envelop above the actual number of requests, and the more accurate the predictions are, the tighter the envelop is.

V. Energy Savings

We now evaluate the performance of our framework in terms of energy savings. We rely on the energy measurements and findings provided in [13] and summarized in Table II. The amount of energy saved under our framework when compared to when there is no power management is calculated in each minute as follows. Let $T_{sleep}(i)$ be the length of the time period during which PM i remains in the sleep mode during the last minute. The energy difference $E_{diff}(i)$ between leaving the PM ON and putting it to sleep is equal to $(P_{idle} - P_{sleep})T_{sleep}(i)$. Letting N_{sleep} denote the set of PMs that were in the sleep mode during the last minute, the total saved energy (in that minute) is $Total\,Saved\,Energy = \sum_{i \in N_{sleep}} E_{diff}(i) - E_{o \to s} - E_{s \to o}$, where $E_{o \to s}$ and $E_{s \to o}$ are the energy overheads of all the switches respectively from the ON to the sleep mode and from the sleep mode to the ON mode during the last minute.

Fig. 7 shows the amount of energy saved when our power management framework is used. For comparison, we also plot the energy savings achieved under the optimal power management, which represents the case when the predictor knows the exact number of future VM requests, as well as the exact amounts of CPU and memory associated with each request (i.e., perfect prediction). This represents the best-case scenario and serves here as an upper bound. The figure shows that our framework achieves great energy savings, and that the amount of saved energy is very close to the optimal case. The gap between our framework and the optimal power saving mode is due to prediction errors and to the redundant PMs that are left ON as a safety margin. Our results indicate that the total amount of saved energy during the entire testing period, calculated by accumulating the energy savings in each minute during the whole testing period, is 18.92 Mega Joules (under our framework) and 19.67 Mega Joules (under optimal power management).

VI. Conclusion

We proposed an integrated energy-aware resource management framework for cloud data centers. Our proposed framework monitors PMs in real time, and effectively decides whether and when PMs need to be put to sleep to save energy. We evaluated our proposed framework through real Google data traces, and showed that it yields great energy savings.

References

[1] R. Brown et al., "Report to congress on server and data center energy efficiency: Public law 109-431," 2008.

[2] "Cisco global cloud index: Frocast and methodology, 2011-2016," *Cicso Inc., White Paper*, 2012.

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

[4] C. Dabrowski and F. Hunt, "Using Markov chain analysis to study dynamic behaviour in large-scale grid systems," in *Proceedings of ACM Symposium on Grid Computing and e-Research*, 2009.

[5] Y. Zhang, W. Sun, and Y. Inoguchi, "CPU load predictions on the computational grid," in *Proceedings of IEEE International Symposium on Cluster Computing and the Grid (CCGRID)*, 2006.

[6] Y. Wu, Y. Yuan, G. Yang, and W. Zheng, "Load prediction using hybrid model for computational grid," in *Proceedings of ACM International Conference on Grid Computing*, 2007.

[7] T. Heath, B. Diniz, E. Carrera, W. Meira Jr, and R. Bianchini, "Energy conservation in heterogeneous server clusters," in *Proc. of ACM SIGPLAN Symp. on Principles and Practice of Parallel Programming*, 2005.

[8] J. Prevost, K. Nagothu, B. Kelley, and M. Jamshidi, "Prediction of cloud data center networks loads using stochastic and neural models," in *Proc. of IEEE Int'l Conf. on System of Systems Engr. (SoSE)*, 2011.

[9] *http://code.google.com/p/googleclusterdata/*.

[10] J. Han, M. Kamber, and J. Pei, "Data mining: concepts and techniques," 2006, Morgan kaufmann.

[11] E. Man Jr, M. Garey, and D. Johnson, "Approximation algorithms for bin packing: A survey," *Journal of Approximation Algorithms for NP-Hard Problems*, pp. 46–93, 1996.

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

[13] I. Sarji, C. Ghali, A. Chehab, and A. Kayssi, "CloudESE: Energy efficiency model for cloud computing environments," in *Proceedings of IEEE International Conference on Energy Aware Computing*, 2011.