Received March 24, 2015, accepted April 25, 2015, date of publication May 6, 2015, date of current version June 1, 2015.

Digital Object Identifier 10.1109/ACCESS.2015.2430276

A Tree Regression-Based Approach for VM Power Metering

CHONGLIN GU, PENGZHOU SHI, SHUAI SHI, HEJIAO HUANG, (Member, IEEE), AND XIAOHUA JIA, (Fellow, IEEE)

Department of Computer Science and Technology, Harbin Institute of Technology Shenzhen Graduate School, Shenzhen 518055, China
Shenzhen Key Laboratory of Internet Information Collaboration, Shenzhen 518055, China

Corresponding author: H. Huang (hjhuang@aliyun.com)

This work was supported in part by the Shenzhen Strategic Emerging Industries Program under Grant JCYJ201-30329153215152, Grant JCYJ20120613151201451, and Grant ZDSY20120613125016389, and in part by the National Natural Science Foundation of China under Grant 11371004.

ABSTRACT Cloud computing is developing so fast that more and more data centers have been built every year. This naturally leads to high-power consumption. Virtual machine (VM) consolidation is the most popular solution based on resource utilization. In fact, much more power can be saved if we know the power consumption of each VM. Therefore, it is significant to measure the power consumption of each VM for green cloud data centers. Since there is no device that can directly measure the power consumption of each VM, modeling methods have been proposed. However, current models are not accurate enough when multi-VMs are competing for resources on the same server. One of the main reasons is that the resource features for modeling are correlated with each other, such as CPU and cache. In this paper, we propose a tree regression-based method to accurately measure the power consumption of VMs on the same host. The merits of this method are that the tree structure will split the data set into partitions, and each is an easy-modeling subset. Experiments show that the average accuracy of our method is about 98% for different types of applications running in VMs.

INDEX TERMS Virtual machine (VM), metering, measure, power, cloud computing.

I. INTRODUCTION

In recent years, many Internet service providers have built their own data centers to process the ever increasing big data. It is estimated that the data centers will consume more than 100 billion kWH per year [1], and the energy cost of data centers will double every 5 years [2]. The most popular solution is to consolidate VMs to as few servers as possible, with remained idle servers shut off. In fact, much more power can be saved if the power consumption of each VM can be accurately measured. Therefore, VM power metering is significant for the power saving of green data centers. Besides, it is reasonable to charge users according to the power consumption of their VMs. The existing service provider like Amazon charges users by configuration types and running time of VMs [3], [4]. The problem lies in that, for VMs with the same configuration and running time, the resource used can be totally different, causing different power consumption.

However, there are several challenges to conquer for VM power metering. VM is running at the level of software, so that traditional hardware power meter cannot be used. It is hard to measure the power consumption of virtual devices like CPU, memory and disk that belongs to a VM. And it is necessary to distinguish the proportion of hardware resources used by each VM.

In this paper, we address the issues mentioned above and propose a tree regression based model for VM power metering. The procedures for VM power metering are as follows: information collection, modeling, evaluating and adjusting. In the architecture of our system, the synchronization for sampling is emphasized. Different from existing work, we regard the modeling features are not independent of each other. Our tree regression model will automatically split the dataset into easy modeling partitions, such that the accuracy problem incurred by the correlation between different resources has no impact on our model. Tree pruning technique has been used for optimization, and it proved to be effective in enhancing accuracy for CPU intensive applications. We also propose a new evaluation approach that reflects the extend of error changes very well in real use. Experiments show that our method is much more accurate than the commonly used models in literature, but as light-weighted as linear model. It also shows that our model is applicable in different

scenarios including CPU intensive, IO intensive, and distributed applications on Hadoop. Above all, our model is suitable for estimating the power consumption of both physical server and VMs running on it.

The rest of paper is organized as follows. In Section II, we summarize the latest literature in VM power metering. Section III presents the architecture of our VM power metering system. In Section IV, we demonstrate our tree regression based modeling method in detail. In Section V, we introduce experiment setup and make evaluations. Finally, Section VI concludes this paper.

II. RELATED WORK

In this section, we will review up-to-date literature regarding VM power metering, including information collection such as methods, tools can be used, as well as sampling interval. We also review various modeling methods in literature as follows.

A. INFORMATION COLLECTION

The modeling information includes two parts: physical server power and profiling features of the resources.

To collect server power, there are two methods: one is to use externally attached PDUs (Power Distributed Unit) like WattsUp series [5] and Scheleifenbauer power meter [6]. The data can be logged inside the PDU or can be accessed through local network area. The other is to use the APIs provided by the server with built-in power meter. For instance, Dell Power Series provide comprehensive power information for each component inside the server through Dell Open Management Suite [7]. PDU is convenient to be attached to and detached from servers, but infeasible in large scale. In contrast, server with inner power meter is preferred for the power management of future data center, though it may bring performance degradation when sampling too frequent. Still, others use wires to connect their self-developed power meter with each component in the server to measure the components' power [8]–[10]. But this method is too complex to be used widely, and Dell Series has already been able to provide power information of major components.

To collect profiling features of resources, the information is collected from CPU, memory, and IO. To account the portion of CPU usage by each VM, Kansal *et al.* [11] propose to transform the tracked performance counters of each VM into the utilization of physical processor. Stoess *et al.* [12] directly use PMCs (Performance Monitor Counters) for each VM. IBM in [13] uses time slices of processors to account the portion of CPU usage by each VM. For memory, Bao *et al.* [14] believe the throughput of memory can well reflect the variation of memory power, while Kansal and Krishnan profiled their memory utilization using LLC missed in [11] and [15]. Still, Kim *et al.* [16] estimate the power consumption of memory using the number of memory accesses. For IO, Kansal uses disk throughput to estimate disk power, while Stoess believes the finishing time of an IO request is more reliable. IBM has implemented monitoring of IO throughput for each VM at hypervisor level of Xen.

It is a complex task to implement the above mentioned methods for modeling information collection. Fortunately, there have been some tools for collecting profiling features of resources at system level, some are designed specifically for profiling VM. Table 1 summarizes the tools that can be used for profiling in virtulization platform.

TABLE 1. Tools for profiling in virtualization platform.

Virtualization	Tools
Xen	XenOprof Xenperf Xentrace Xentop Xenanalyze Xenstat XenMon
KVM	Perf Suite Oprofile Perfmon2
VMWare	ReTrace vmkperf

In information collection, the rate of sampling should also be taken into consideration. Sampling too frequent will incur degradation of performance; otherwise, the modeling accuracy will decline. An empirically setting for sampling rate is 1∼2 seconds [17], [18]. In fact, the sampling rate should be adjusted according to the variation of running applications, as is mentioned in [19]. In our expriment, we choose 2 seconds as our sampling rate.

B. MODELING METHODS

From the perspective of information collection, VM power metering can be classified into two categories: white box method and black box method. As is mentioned in [20], white box method collects the modeling information of each VM using a proxy running inside of each VM. This method breaks the integrity of VM, not suitable for providing VM services like Amazon EC2. It is inaccurate when modeling information is collected from different VM OSes such as Windows and Linux. In view of this, most recent researches use black box method that collect the modeling information of each VM at the level of hypervisor or host.

From the perspective of modeling method, VM power metering can be classified into linear model and non-linear model. Let the total server power denoted as P_{Total}, static power denoted as P_{Static}. The general model for both linear and non-linear is based on the assumption that the total server power consists of static power and the power consumed by each component such as CPU, memory, and IO, denoted as P_{CPU}, P_{Mem}, P_{IO}, respectively. Thus, we have:

$$P_{Total} = P_{Static} + P_{CPU} + P_{Mem} + P_{IO}$$

In linear model, the power consumption of each component is in linear relationship with utilization of the resource, denoted as R_{CPU}, R_{Mem}, and R_{IO} for CPU, memory and IO, respectively. Thus, we have:

$$P_{Total} = P_{Static} + \alpha R_{CPU} + \beta R_{Mem} + \gamma R_{IO} + e,$$

where e represents the adjusting variable in linear regression. Since the power consumption of each component is caused by running VMs, then:

$$P_{Total} = P_{static} + \alpha \sum_{i}^{n} R_{VM_i}^{CPU} + \beta \sum_{i}^{n} R_{VM_i}^{Mem} + \gamma \sum_{i}^{n} R_{VM_i}^{IO} + e,$$

where $R_{VM_i}^{CPU}$, $R_{VM_i}^{Mem}$, and $R_{VM_i}^{IO}$ denote CPU, memory, and IO usage by each VM_i, respectively. α, β, and γ are the parameters to be trained. Thus, the power consumption of each VM i, denoted as P_{VM_i}, is:

$$P_{VM_i} = \alpha R_{VM_i}^{CPU} + \beta R_{VM_i}^{Mem} + \gamma R_{VM_i}^{IO}.$$

Among the linear models, Kansal *et al.* [11] use CPU utilization, LLCM (Last Level Cache Missing), and transfer time of IO for modeling. Krishnan *et al.* [15] only use instructions retired and LLC (Last Level Cache) hits for his linear model. Kim *et al.* [16] considered the number of active cores, retired instructions, and number of memory accesses in his linear model. Similarly, Bertran *et al.* [21], [22] also considered the number of active cores for his linear model. Chen *et al.* [19] propose a modified model using CPU and Hard disk. Bohra and Chaudhary [23] use PMCs such as CPU_CLK_UNHALTED, DRAM_ACCESSES, INSTRUCTION_CACHE_FETCHES and DATA_CACHE_FETCHES to represent the component states of CPU, memory and caches for modeling. The only difference among those linear models is the component slection for modeling. In linear models, least squares is often used for multi-variable linear regression.

As for non-linear models, methods can be classified into polynomial models, lookup table method, and machine learning models. For polynomial modeling method, Versick *et al.* [24] and Waßmann *et al.* [25] propose a polynomial formula, and the maximum accuracy can be reached when the polinomial order is six. Xiao *et al.* [26] and Peng and Sai [27] build his polynomial model using PMCs. For lookup table method, Jiang *et al.* [28] just build an matrix called LUT to store the CPU and LLC, fill the matrix with collected data and interpolate data by the designed rule. But the table is too large to be retrieved when more features are considered. For machine learning modeling method, Yang *et al.* [18] adopt a machine learning method called ε-SVR model for VM power metering.

Other work related to VM power metering are: Liu *et al.* [29] build a model for estimating the power consumption of VM migration, and he finds that the migration power is in linear relationship with transmitting volumes. Ma *et al.* [30] take the power consumption of fan into consideration for each VM. Quesnel *et al.* [31] propose a method that fairly divides the idle server power into each VM.

To sum up, linear model is a commonly used method in VM power metering for its simplicity in implementation, with low overhead when running. However, it is built on the assumption that all the input variables are independent of each other [17]. It is obvious that the parameters should be trained frequently when the behaviors of applications always vary, causing high overhead. Non-linear model may improve the accuracy to a certain extend, but too complex especially in updating parameters. In this paper, we propose a tree regression based method for VM power metering. It is a simple method with high accuracy and low overhead as linear model. The following sections will demonstrate the architecture of our system in detail.

III. SYSTEM ARCHITECTURE

In this section, we will describe the basic architecture of our system for VM power metering, as is shown in Figure 1. This architecture is based on black box method that collects modeling information of each VM at the level of host.

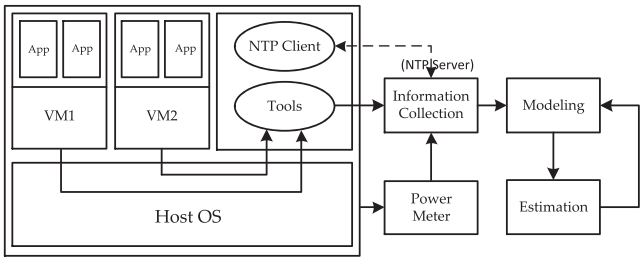


FIGURE 1. Architecture of VM Power Metering.

In this architecture, several VMs are running on the host, each with several applications running inside. The first step for VM power metering is information collection, so there are tools collecting the features of each VM at host level. A separate server is running for gathering the modeling information of the host server and the information from PDU. It is worth emphasizing that the information collecting server runs a NTP service for synchronizing the timestamps of resource information and power information. The second step is modeling, and there is a modeling module specifically responsible for training parameters based on collected samples. The last step is to evaluate the accuracy by calculating the error between estimated and measured server power. The estimation module is also responsible for updating parameters when errors exceed a certain threshold. With all these modules, our system can provide high quality service for VM power metering in real application.

IV. TREE REGRESSION BASED METHOD FOR VM POWER METERING

For VM power metering, the commonly used linear fitting method is inaccurate when the interrelations of the resource features of the training data are too complex. In fact, the resource features are not always in rigor linear relationship. The pattern of resource features with power may vary for different applications. Thus, modeling should be closely related to collected dataset for this application, and a separate set

of parameters will be fitted on it. One feasible solution is to split the dataset into several easy-modeling subsets. For any subset, it will be splitted until any subset fits the model very well. This is a regression problem, and it can be formulated like this: suppose there is a training data of n observations, each has a response variable y, and a vector of predictor variables x with length m. The domain of x and y are denoted as X and Y, respectively. Thus, the training samples can be denoted as $D = \{(x_1, y_1), \ldots, (x_n, y_n)\}$. Our goal is to find a proper model for estimating y from a new predictor x. In theory, the solution is to split X into k disjoint pieces, $A_1, A_2, \ldots, A_k$, such that $X = \bigcup_{i=1}^{k} A_i$ and the mean squared prediction error $(y_{predict} - y)^2$ is minimized [32].

In this paper, we propose a tree regression based approach to measure the power consumption of VMs. It recursively partition the dataset into several easy modeling pieces. For tree regression method, it usually can be classified into model tree and regression tree. The only difference between these two types of trees is that the former uses linear regression to fit the partitioned data, while the latter uses a constant for each partitioned data. We will evaluate tree regression methods in Section V.

There are basically three steps for VM power metering:

Step 1: Information collection, including static server power, profiling features of resource for both server and VM, and real server power when VM runs tasks inside.

Step 2: Build a tree model for server power. It involves two algorithms: CreateTree and FeatureValueSelect.

Step 3: Apply the built tree to estimate the power consumption of each VM.

In tree building, there are four major tasks for VM power metering: (i) how to create a tree, (ii) how to make partitions for training data, (iii) when to stop partitioning, (iv) how to apply our tree to estimate the power consumption of VM. Since model tree has been proved to perform well in our experiment, we will mainly discuss our tree regression method using model tree to address the above mentioned four tasks in the following.

A. BASIC THEORY

We build our tree using resource features including CPU utilization, Last Level Cache Misses (LLCM), bytes of throughput for VM power metering. All these features have been proved to be effective in power modeling [11], [15], [29]. We collect the information for each VM using blackbox method. Thus, the accuracy of server power model will have great impact on the the estimated VM power [20].

Here are some definitions. Suppose there are n observations in the training dataset. The response variable of each observation is server power, denoted as P_{Total}. The predictor variables of each observation is a vector of resource usage, denoted as R. $R = \{R_{CPU}, R_{memory}, R_{IO}\}$, where R_{CPU}, R_{memory}, and R_{IO} denote CPU utilization, LLCM, and IO throughput, respectively. Thus, the training samples can be denoted as $D = \{(R_1, P_1), \ldots, (R_n, P_n)\}$. The estimated power of server using tree regression method can be denoted as $P_{Estimated}(d), d \in D$. For each non-leaf node (also called partitioning node) in the tree, denoted as *Node*, it has four fields, (*feature*, *value*, *right*, *left*). *feature* represents resource feature such as CPU, memory, or IO, *value* is a real value of the resource *feature*. *right* and *left* represent the right and left child of this node, respectively. For leaf node, it has no fields for left child and right child. It has only two field, *feature* and *value*. *feature* is null if it is a leaf node. *value* is not a real number for leaf node. Instead, it has two fields: *data* and *paras*, denoting the partitioned dataset of the leaf and the modeling parameters fitted on this dataset, respectively. In fact, the *data* field is not necessary in real use if we do not optimize the tree by punning. In this paper, we will discuss pruning using the partitioned dataset of the leaf node, so that leaf node is designed with two fields. For ease of understanding, Figure 2 is a simple example to illustrate the data structure for partitioning node and leaf node of our tree:

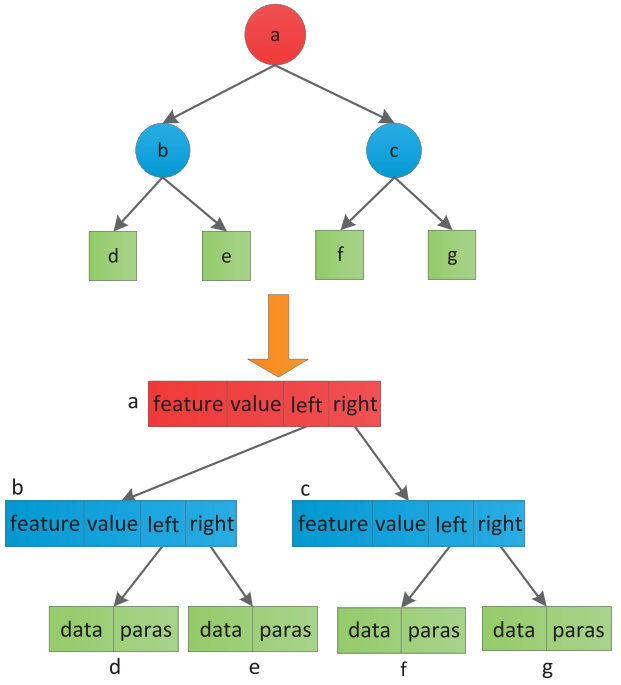


FIGURE 2. A Simple Example of Our Tree Structure.

B. CREATING TREE

The main idea of CreateTree is to recursively partition the dataset into subsets until there is no proper features to be selected for partitioning. In each partitioning, a proper (*feature*, *value*) pair is selected in the dataset. The partition happens only when the enhancement of accuracy exceeds a certain threshold. If the selected *feature* is null, the input dataset with modeling parameters fitted on it will be added to the tree as a leaf node, like that in Figure 2.

In Algorithm 1, function SplitData is to partition the training dataset D into two parts $D1$ and $D2$ by the *value* of resource *feature*. It is worth noting that function

Algorithm 1 CreateTree

Input:
The training dataset D;
Output:
A tree T;
1: $(feature, value) = FeatureValueSelect(D, s, t)$;
2: **if** $feature = null$ **then**
3: **return** $value$;
4: **else**
5: $T.feature = feature$;
6: $T.value = value$;
7: $(D_1, D_2) = SplitData(D, feature, value)$;
8: $T.left = CreateTree(D_1)$;
9: $T.right = CreateTree(D_2)$;
10: **end if**

FeatureValueSelect is the key in tree creating. It always selects the best (*feature*, *value*) pair for further partitioning. The following will give FeatureValueSelect algorithm.

C. FEATUREVALUESELECT

The main idea of FeatureValueSelect (see Algorithm 2) is to select the best (*feature*, *value*) pair for partitioning in tree creation. In fact, the selected *value* is obtained through trying of partitions using the values *feature* from samples of the input dataset one by one. As a result, the error of this tree can be reduced after partitioning. The inputs of this algorithm includes two stopping conditions: s and t, denoting minimum size of partitioned dataset and threshold of error reduction, respectively. If any partitioned subset is smaller than s in size, or the error reduction after partitioning is less than t, this trying is invalid, and this partition will not really happen in creating tree. The two thresholds s and t avoid unnecessary partitions in enhancing accuracy, so that the tree can be created in a much faster speed. The *feature* is null if no proper feature can be found in the dataset, or threshold conditions cannot be satisfied.

In Algorithm 2, *Error* function quantifies the overall deviation between measured server power and estimated power. It is calculated as follows:

$$Error(D) = \sum_{i}^{n} (d_i \cdot P_{Total} - P_{Estimated}(d_i))^2, \quad d_i \in D,$$

where $P_{Estimated}(d_i)$ is the estimated power. In fact, *MakeLeaf* function is to generate a leaf node, the output of this function includes two fields: input dataset and parameters fitted on it using linear model. For regression tree, the leaf node contains (*data*, C), where C is the mean value of response variables in the dataset.

Traditional linear resource-power model needs to fit all the collected samples globally with relatively high error for some dataset. So Li *et al.* [33] propose a piecewise linear fitting model. But the method is too subjective in partitioning into high, middle and low utilization for a certain feature. In fact, our tree regression is also a piecewise modeling method.

Algorithm 2 FeatureValueSelect

Input:
The training dataset D;
Minimum partition size s;
Error reduction threshold t;
Output:
(*bestfeature*, *bestvalue*);
1: $olderr = Error(D)$;
2: $newerr = 0$;
3: $besterr = Infinit$;
4: $bestfeature = nulls$;
5: $bestvalue = nulls$;
6: **for all** $d \in D$ **do**
7: **for all** $feature \in \{CPU, memory, IO\}$ **do**
8: $(D_1, D_2) = SplitData(D, feature, d.R_{feature})$;
9: **if** $size(D_1) < s$ or $size(D_2) < s$ **then**
10: continue;
11: **end if**
12: $newerr = Error(D_1) + Error(D_2)$;
13: **if** $newerr < besterr$ **then**
14: $besterr = newerr$;
15: $bestfeature = feature$;
16: $bestvalue = d.R_{feature}$;
17: **end if**
18: **end for**
19: **end for**
20: **if** $0 < olderr - besterr < t$ **then**
21: **return** $(null, MakeLeaf(D))$;
22: **end if**
23: **if** $size(D_1) < s$ or $size(D_2) < s$ **then**
24: **return** $(null, MakeLeaf(D))$;
25: **end if**
26: **return** $(bestfeature, bestvalue)$;

But our partition is executed automatically based on training data, rather than artificial partitioning. Thus, the generated model, especially partition, will fit the dataset much better.

D. TREE PRUNING

During the process of building tree, each partition will generate more branches and leaves. In pruning, a separate test dataset will be used. It recursively merges the leaves of this tree until further merging will not enhance the accuracy of this tree. Through pruning, all over-fitting partitions will be merged into one, and the merging is executed recursively starting from leaf nodes.

In Algorithm 3, function *Istree* is to judge if the input is a tree or a leaf node. Function *Merge* is to merge the datasets of two leaves with the same parent into one dataset. Function *Linear* is to train parameters based on the input dataset. Function *Evaluate* is to calculate quadratic sum of deviations between measured and estimated power for the dataset using the input modeling parameters. The leaves of the same parent will be merged if the accuracy after merging can be enhanced. Tree pruning has been proved to be a useful

Algorithm 3 TreePruning

Input:
Tree T;
Test Dataset D;
Output:
A tree T;

if $Istree(T.left)$ or $Istree(T.right)$ **then**
  $(D_1, D_2) = SplitData(D, T.feature, T.value)$;
end if
if $Istree(T.left)$ **then**
  $T.left = TreePruning(T.left, D_1)$;
end if
if $Istree(T.right)$ **then**
  $T.right = TreePruning(T.right, D_2)$;
end if
if not $Istree(T.left)$ and not $Istree(T.right)$ **then**
  $(D_1, D_2) = SplitData(D, T.feature, T.value)$;
  $ErrorNoMerge = Evaluate(D_1, T.left.paras) + Evaluate(D_1, T.right.paras)$;
  $MergeData = Merge(T.left.data, T.right.data)$;
  $Mergeparas = Linear(MergeData)$;
  $ErrorMerge = Evaluate(D, Mergeparas)$;
  if $ErrorMerge < ErrorNoMerge$ **then**
    return $(MergeData, Mergeparas)$
  else
    return T;
  end if
else
  return T;
end if

optimization for CPU intensive applications, which will be discussed in Section V.

E. ESTIMATION OF POWER CONSUMPTION

For VM power metering, the first step is to build a proper model for server power. The following algorithm 4 describes how to estimate the power consumption of server using built regression tree. And then, the power consumption of each VM can be calculated easily.

In Algorithm 4, there is always a proper leaf can be found in the regression tree for any new observation, and its power can be estimated using the parameters in this leaf. Function *Calculate* is to calculate the power consumption for this observation. For each partitioned dataset, a linear model is fitted on it and added to the tree as a leaf node. So we have:

$$P_{Total} = P_{Static} + \alpha R_{CPU} + \beta R_{LLCM} + \gamma R_{IO} + e,$$

In this paper, we collect the profiling features of each VM at host level. The resource features for each VM is contained in that of the server. Thus, VM power can be divided from host power in a fair way, so we have:

$$P_{VM_i} = \alpha R^{CPU}_{VM_i} + \beta R^{Mem}_{VM_i} + \gamma R^{IO}_{VM_i} + e_i,$$

Algorithm 4 Estimation of Server Power

Input:
New observation $d = (R, P_{real})$;
$R = \{R_{CPU}, R_{memory}, R_{IO}\}$;
A tree T;
Output:
$P_{Estimated}(d)$;

while T is not leaf **do**
  if $d.R_{feature} < T.value$ **then**
    $T = T.left$;
  else
    $T = T.right$;
  end if
end while
$P_{Estimated}(d) = Calculate(d.R, T.paras)$;
return $P_{Estimated}(d)$;

where e_i is the bias for VM_i, and

$$e_i = e * (\alpha R^{CPU}_{VM_i} + \beta R^{Mem}_{VM_i} + \gamma R^{IO}_{VM_i} + e_i) / \sum_{j=1}^{n} (\alpha R^{CPU}_{VM_j} + \beta R^{Mem}_{VM_j} + \gamma R^{IO}_{VM_j})$$

F. ACCURACY EVALUATION

For VM power metering, evaluating the accuracy is very important. The parameters adjusting will heavily rely on the evaluation method. It has been acknowledged that a good model for server power will always give predictions with limited error [11]. If we have an accurate server power model, then VM power metering can be transformed into a problem of how to fairly divide server power into each VM. The evaluation in literature is usually given as follows:

$$Error\_rate = \frac{1}{n} \sum_{i=1}^{n} \frac{|P_{Estimated}(d_i) - d_i.P_{Total}|}{d_i.P_{Total}}, \quad (1)$$

where $P_{Estimated}$ denotes the estimated server power.

Although this evaluation method has been used widely, there exists a serious problem. As is known to all, the power of a server consists of dynamic power, denoted as $P_{dynamic}$ and static power, denoted as P_{static}

$$P_{Total} = P_{dynamic} + P_{static}$$

When P_{static} is too large, the *Error_rate* will be reduced to be very small. Thus, the evaluation result is not satisfying, even if it is less than 5%. For example, suppose the basic power for a server is 200W, when the real error is 10W, the *error_rate* will be 5%. Formula (1) is the most popular used method in literature, but it does not reflect the real error very well.

To be more objective in evaluation, the basic server power as well as deviation between $P_{Estimated}$ and P_{real} should be given. Thus, we can know better how the system performs in power estimation. In this paper, we propose a new method to

evaluate the accuracy of power models, and the formula is:

$$Error\_rate = \frac{1}{n}\sum_{i=1}^{n}\frac{|P_{Estimated}(d_i) - d_i \cdot P_{Total}|}{d_i \cdot P_{dynamic}} \quad (2)$$

It reflects the extend of errors against dynamic power more objectively. We use our proposed method to make evaluation. For comparison with work in literature, we also gives our accuracy using formula (1).

Using our tree regression model, we do not need to update parameters too often. Because any new observed data will always find a proper set of modeling parameters by searching the leaf node in the tree. And each leaf node is separately modeled using its partitioned dataset. In Algorithm 2, two threshold such as size s and error t are considered. The best s and t are selected by constant trying. Even if we got the best parameters, we do not ensure current tree still performs best for testing dataset. So tree pruning, as an optimization technique, will be tried in our experiment.

V. EXPERIMENTS

In this section, we will introduce the details of our experiment platform including the server configuration, tools used, benchmarks. Through result analyzing, we find our tree regression method for VM power metering is very promising.

A. EXPERIMENTS SETUP

The configuration of server in our experiment is Lenovo T350 with 4 cores of CPU and 8G memory. The static power of our server is 166 Watts, which is collected when server is idle. There are three VMs running on the host, each is dispatched with one two-core CPU, 2GB memory. The information collection server is a HP PC with 2GB memory. It also runs NTP service. All the OSes of server and PC are Ubuntu. We use KVM as virtualization platform for our experiment, due to its light-weighted merit compared with other virtualization solutions. Each VM will run as a separate process in system, so that tools like *perf* is adopted for profiling each VM. We also use *sysstat* and *iostat* for profiling other modeling related information. The power meter is an externally attached PDU called HOPI 9800 made in China. It provides real time power information such as energy consumption (KWH), power (Watts), voltage (Volt), current (Amp), and its minimum resolution is 0.001 Watt. All the information can be accessed in real time through USB connection to PC.

To evaluate our method, we choose some classical benchmarks including CPU intensive, IO intensive, and distributed hadoop benchmarks, as are listed in Table 2.

For UnixBench, we choose 8 benchmarks as our testing instances in Table 2. We run the instances one after another, and the training data is the combination of all samples of those benchmarks. One reason is that a well-built model should be based on sufficient samplings, versatile instances are needed. Another reason is that the running time for each of the testing instance in UnixBench is too short, and the size of dataset is not large enough for a good modeling. For IOZone benchmark, by adjusting the throughput of reading and writing, the observations of this benchmark is sufficient for modeling. As for Hadoop, there are three benchmarks such as pi, sort and write. For each benchmark, three VMs cooperate to finish the task. And each of the benchmarks can generate a single dataset by adjusting its workload. Thus, we have five datasets in total. Note that the power consumption does not solely depend on the size of input, it also rely on the inner computation of this application. Through adjusting parameters of input when running benchmarks, we just try to collect sufficient observations for modeling.

TABLE 2. Benchmarks.

Benchmarks	Test Instances	Descriptions
UnixBench	dhry2 whets hanoi int short long float double	CPU Intensive Tests
IOZone	read write	IO Intensive Tests
Hadoop	pi sort randomwriter	Distributed System Tests

In experiment, each dataset is divided into two parts, training set and testing set, which account for 99% and 1%, respectively. In tree pruning, a separate subset is chosen from training set for optimization, and it has the same size as testing set. Why not dividing the whole dataset empirically into subsets of 90% and 10% in proportion? The reason is that when dataset is not so large, this way of dividing will cause more errors, especially for our tree model. So we try to build our tree using as many observations as possible. Since our datasets are not very large, we ingeniously use K-fold cross validation method to evaluate our model, and K is set to be 100. In other words, there are 100 different groups of datasets, each group has two parts of 99:1 in proportion to the whole dataset. Thus, we can objectively evaluate our model without sacrificing the accuracy of our method when modeling.

B. RESULT ANALYSIS

1) PARAMETERS SELECTION

Before analyzing the final result regarding the accuracy of our model, parameters selection for s and t should be discussed first. s is the minimum size of subset that should be satisfied in partitioning. And t is the minimum error reduction that should be satisfied after partitioning. The first experiment we have done is to select the best parameters for our tree. For each combination of s and t, 100-fold cross validation has been used, so that the final result can be evaluated in an objective way. Here are the best parameters selected on different benchmarks, as are shown in Figure 3 and Figure 4.

Figure 3 is the best *size* for regression tree and model tree on different benchmarks. It can be easily found that the best *size* in regression tree is much smaller than that of model

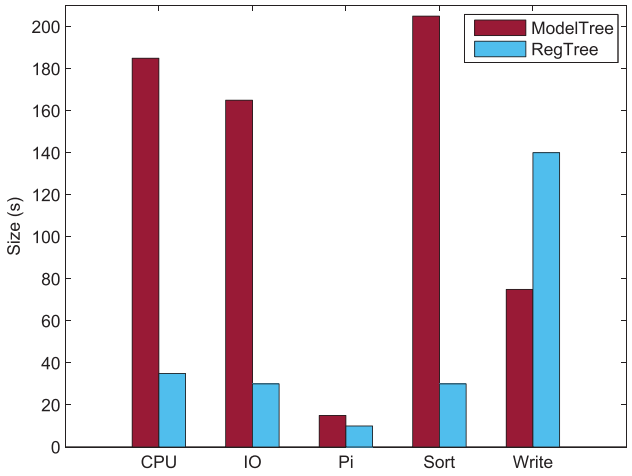


FIGURE 3. Best Size for RegTree and ModelTree.

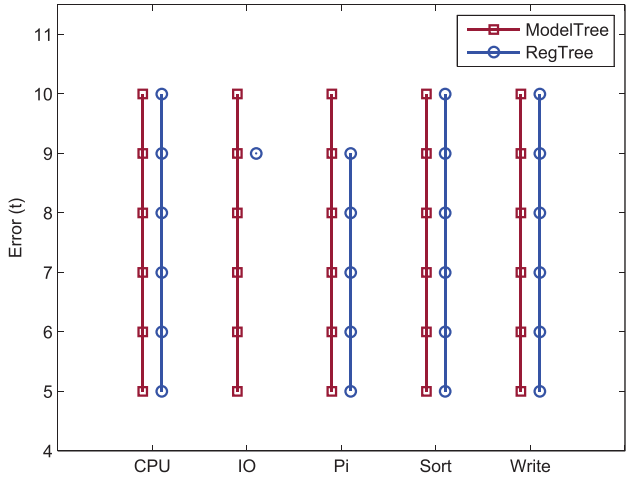


FIGURE 4. Best Error Threshold for RegTree and ModelTree.

tree except for the last benchmark. For the regression tree, each leaf model is the mean of the response variables for the dataset, too large leaf will lead to large variance unless each partitioned dataset has similar values like that of benchmark Write. Even though, better model can be fitted with larger partitioning size for model tree in general.

Figure 4 is the best error threshold for regression tree and model tree on different benchmarks. Through experiment, we have found that, if the error threshold is set too small such as 1 and 2, it is too tight to make partitioning, such that the advantages of tree regression can not be made full of. It will also cost a lot of time in creating tree, and the accuracy of built tree is not very high. So we relax the range of t to be 5∼10. Thus, the tree can be built much faster with high accuracy. We found that there are many best pairs of $s-t$ combinations for each benchmark. For example, t can be set to be any from the range 5∼10 with the unique best size s for both model tree and regression tree on benchmark CPU, as is shown in Figure 4. And there is only one best t for regression tree on benchmark IO, and four best t on benchmark Pi. In general, it takes more efforts for selecting the best s compared with t in creating tree.

It is important to find how the changes of s and t affect tree structure such as the number of leaves and depth, and accuracy. Figure 5 shows the variation of the number of leaves, the depth of tree, and accuracy with error t changes when size s is fixed. In this experiment, we fix the size to be 120 and 30 for model tree and regression tree, respectively. Similarly, Figure 6 shows the trends with size s when t is fixed to be 9.

From (a(1), b(1), c(1)) in Figure 5, it can be seen that the number of leaves, depth, and accuracy are unchanging with error threshold t changes on different benchmarks for model tree. While for regression tree, the number of leaves drops slightly on CPU and IO benchmark. But the changing of error threshold has hardly affected accuracy, as is shown in (a(2), b(2), c(2)). Therefore, the threshold of error t has very little impact on leaves, depth and accuracy when size is fixed.

In Figure 6, it can be seen that the number of leaves and depth drop when size increases for both model tree and regression tree. Larger leaf node means shorter height of the tree. In general, the accuracy of both model tree and regression tree will first increase until to its best size, and then the accuracy will decline afterwards. Although there is fluctuations for benchmark Pi, but the overall trend is the same with others.

It is not difficult to conclude that the changing of t has very little effect on both regression tree and model tree. But the changing of s will greatly affect the tree structure such as leaf number and depth, as well as accuracy. In most cases, the best s for model tree is larger than that for regression tree.

2) ACCURACY ANALYSIS

To evaluate the accuracy of our tree regression method, we calculate the error rate using formula (1) and formula (2), denoted as **Evaluation1** and **Evaluation2** in Figure 7. For each benchmark, we run five algorithms, and they are linear regression method, regression tree, regression tree with Optimization (pruning), model tree, and model tree with pruning, short as Line, RT, RTO, MT, MTO in this Figure.

In Figure 7, it can be easily seen that model tree is the most accurate method in general. Tree pruning enhances the accuracy of model tree on CPU benchmark which includes 8 cases, as is shown in (a2). Although pruning does not always ensure accuracy enhancement, it can be a good choice for conditions when our model tree is built with too many leaves with small size, or running computing intensive applications. In Figure 7, regression tree performs bad in general compared with model tree, though it performs best on Hadoop Write. The accuracy of regression tree is unstable regardless of pruning or not, even lower than traditional linear model in some cases. Therefore, model tree is preferred in modeling using tree structure.

In Figure 7, Evaluation2 is using our proposed method, as is shown in formula (2). The accuracy among different algorithms can be distinguished easily using our method.

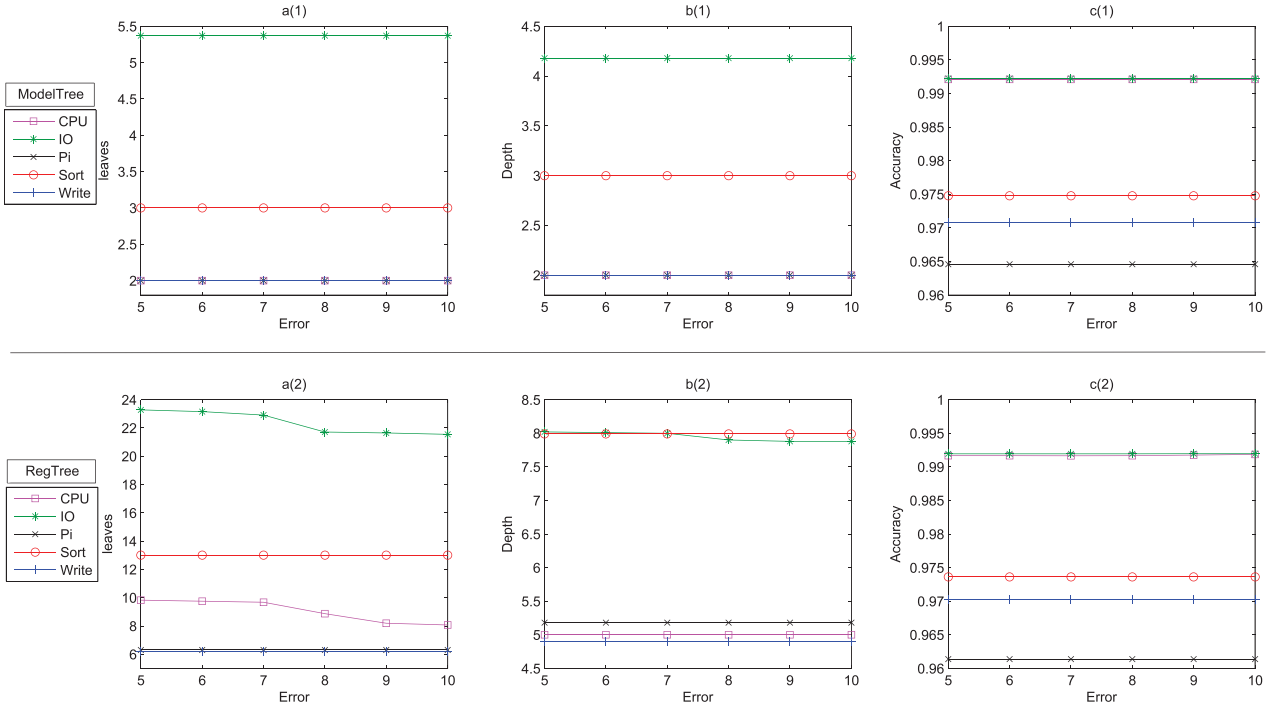


FIGURE 5. The trend of leaves, depth, and accuracy with error threshold when size is fixed. (a1) The number of leaves with error for model tree. (a2) The number of leaves with error for regression tree. (b1) The depth of tree with error for model tree. (b2) The depth of tree with error for regression tree. (c1) The accuracy with error for model tree. (c2) The accuracy with error for regression tree.

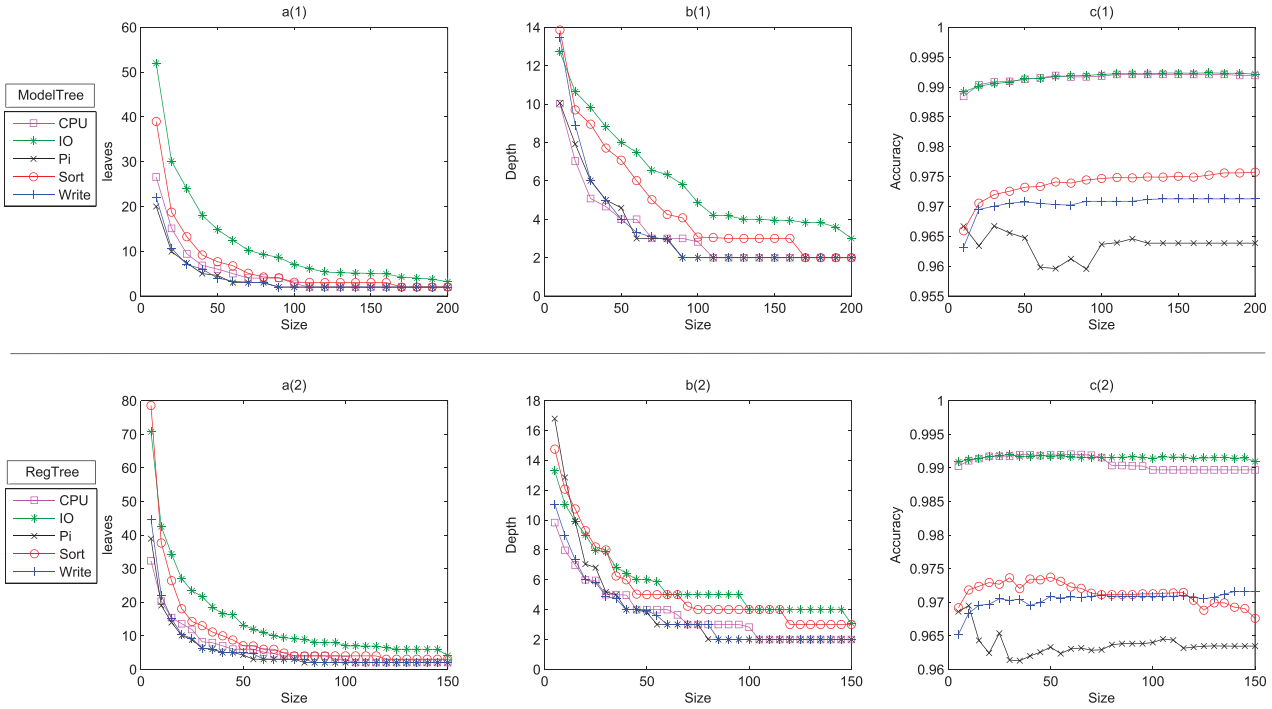


FIGURE 6. The trend of leaves, depth, and accuracy with size when error threshold is fixed. (a1) The number of leaves with size for model tree. (a2) The number of leaves with size for regression tree. (b1) The depth of tree with size for model tree. (b2) The depth of tree with size for regression tree. (c1) The accuracy with size for model tree. (c2) The accuracy with size for regression tree.

For example, the excellence of model tree using optimization (MTO) is much more obvious in a(2) compared with that in (a1). The reason is that formula (1) always has a big P_{Static} added to denominator, so that accuracy of the algorithms seem to be too similar. By comparisons between Evaluation1 and Evaluation2, it can be found that our proposed evaluation method can reflect the extend of error against dynamic power in a more objective and obvious way.

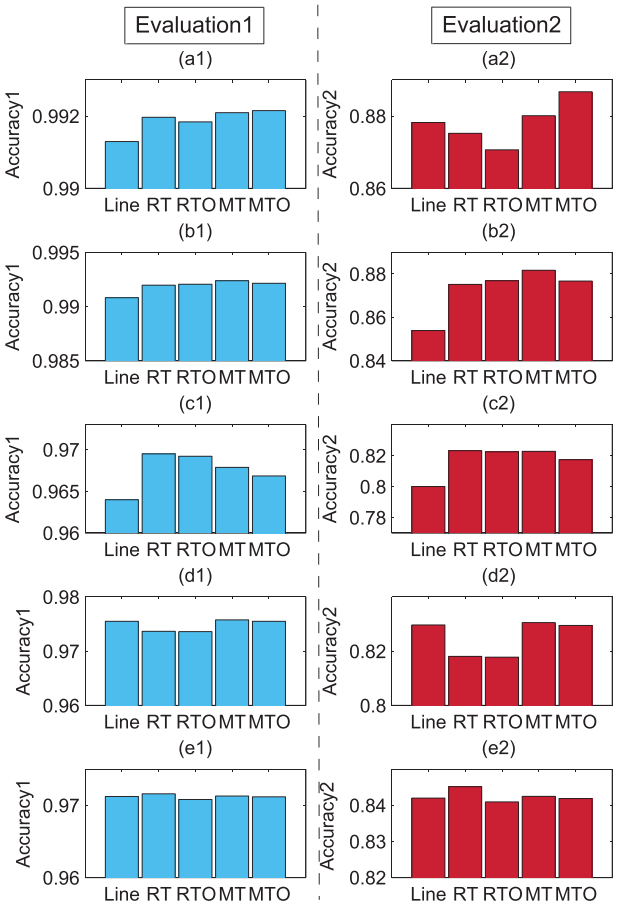


FIGURE 7. Accuracy Evaluations of the Algorithms. (a1) CPU benchmark. (a2) CPU benchmark. (b1) IO benchmark. (b2) IO benchmark. (c1) Hadoop Pi. (c2) Hadoop Pi. (d1) Hadoop sort. (d2) Hadoop sort. (e1) Hadoop write. (e2) Hadoop write.

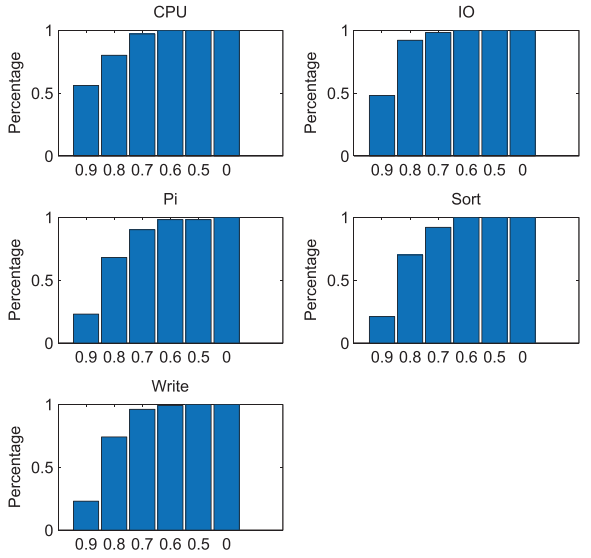


FIGURE 8. Accuracy Distribution of the 100-fold Cross Validation Datasets.

Our model tree gives a satisfying accuracy in experiments. It is necessary to analyze the distribution of accuracy for the 100 groups of datasets in our K-fold cross validation on different benchmarks, as is shown in Figure 8. The accuracy in this Figure is calculated using our proposed formula (2). We found that the proportion of data with accuracy less than 70% is almost none, and most observations are more than 80%. Quite a certain of observations have accuracy more than 90%.

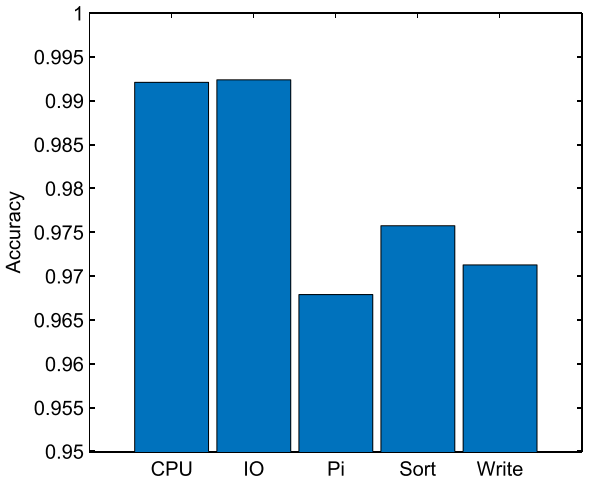


FIGURE 9. Accuracy of Model Tree using Traditional Evaluation Method.

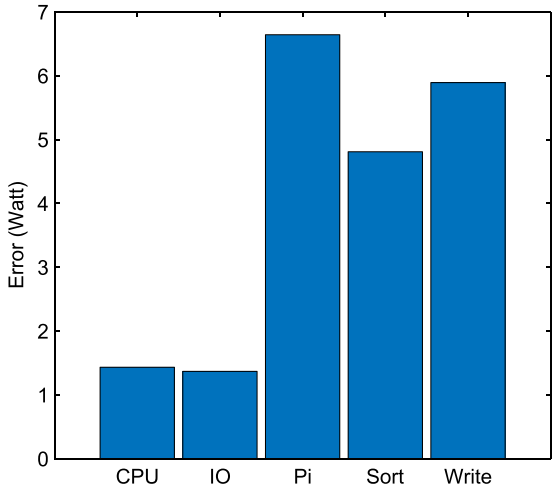


FIGURE 10. Real Errors of Model Tree on Benchmarks.

To evaluate our model tree, we also use traditional evaluation method, denoted as formula (1) in Section IV. In Figure 9, it shows the accuracy of our model tree is almost 98% on average for the five benchmarks, and each is with accuracy more than 96%.

For real error between estimated and measured power, our model tree is very accurate on CPU and IO benchmarks with mean error between 1.3∼1.5 Watts, better than that on Hadoop distributed benchmarks. In Figure 10, the average error of those benchmarks is 4.03 Watts. The relative inaccuracy of Hadoop benchmarks is mainly caused by resource competition among the VMs and the virtualization management from OS. Nonetheless, it proved that out model tree is state-of-the-art for its merits in modeling using tree structure.

VI. CONCLUSION

VM power metering is significant for the power management of green cloud data centers. In this paper, we propose a tree regression based method for VM power metering. It recursively divides the input data into two subsets according to the selected resource feature with value from the dataset. And each leaf node is a data structure with two fields: partitioned dataset, and parameters of linear model fitted on this dataset. Using the observations of server, we build a model tree to estimate the power consumption of server. And then the power consumption of each VM can be obtained by fairly dividing from server power. In parameters selection, we also found that the partition size affects the accuracy of the tree greatly, while error threshold affects little. In addition, a new evaluation method has been proposed to objectively reflect the extend of error changes. Experiments show that our model tree is accurate on different benchmarks, suitable for real use.

Based on our research, there will be more machine learning methods to be applied for VM power metering. One interesting direction is to build a proper black-box models such as neural network models using collected dataset, not just collecting data in black-box way for each VM. Another direction is how to evaluate the accuracy in the granularity of each VM.

REFERENCES

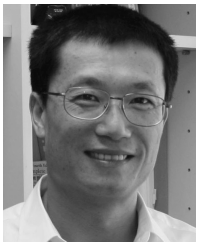
[1] A. Hooper, "Green computing," *Commun. ACM*, vol. 51, no. 10, pp. 11–13, 2008.
[2] R. Brown, "Report to congress on server and data center energy efficiency: Public law 109–431," Lawrence Berkeley Nat. Lab., Berkeley, CA, USA, Tech. Rep., 2008, pp. 1–137.
[3] E. Elmroth, F. G. Marquez, D. Henriksson, and D. P. Ferrera, "Accounting and billing for federated cloud infrastructures," in *Proc. 8th Int. Conf. Grid Cooperat. Comput. (GCC)*, Aug. 2009, pp. 268–275.
[4] M. Armbrust *et al.*, "A view of cloud computing," *Commun. ACM*, vol. 53, no. 4, pp. 50–58, Apr. 2010.
[5] *Watts Up Meter*. [Online]. Available: https://www.wattsupmeters.com/secure/index.php, accessed May 29, 2014.
[6] *Public APIs of Schleifenbauer PDU*. [Online]. Available: http://sdc.sourceforge.net/index.htm, accessed May 29, 2014.
[7] J. Jenne, V. Nijhawan, and R. Hormuth. *Dell Energy Smart Architecture (DESA) for 11G Rack and Tower Servers*. [Online]. Available: http://www.dell.com, accessed 2009.
[8] W. Dargie and A. Schill, "Analysis of the power and hardware resource consumption of servers under different load balancing policies," in *Proc. IEEE 5th Int. Conf. Cloud Comput. (CLOUD)*, Jun. 2012, pp. 772–778.
[9] E. N. Elnozahy, M. Kistler, and R. Rajamony, "Energy-efficient server clusters," in *Power-Aware Computer Systems*. New York, NY, USA: Springer-Verlag, 2003, pp. 179–197.
[10] D. Economou, S. Rivoire, C. Kozyrakis, and P. Ranganathan, "Full-system power analysis and modeling for server environments," in *Proc. Workshop Modeling Benchmarking Simulation (MOBS)*, Boston, MA, USA, 2006, pp. 13–23.
[11] A. Kansal, F. Zhao, J. Liu, N. Kothari, and A. A. Bhattacharya, "Virtual machine power metering and provisioning," in *Proc. 1st ACM Symp. Cloud Comput.*, 2010, pp. 39–50.
[12] J. Stoess, C. Lang, and F. Bellosa, "Energy management for hypervisor-based virtual machines," in *Proc. USENIX Annu. Tech. Conf.*, 2007, pp. 1–14.
[13] Y. Chen, L. Li, L. Liu, H. Wang, and Y. Zhou, "Method and apparatus for estimating virtual machine energy consumption," U.S. Patent 0 296 585 A1, Nov. 22, 2012.
[14] Y. Bao *et al.*, "HMTT: A platform independent full-system memory trace monitoring system," *ACM SIGMETRICS Perform. Eval. Rev.*, vol. 36, no. 1, pp. 229–240, Jun. 2008.
[15] B. Krishnan, H. Amur, A. Gavrilovska, and K. Schwan, "VM power metering: Feasibility and challenges," *ACM SIGMETRICS Perform. Eval. Rev.*, vol. 38, no. 3, pp. 56–60, Dec. 2011.
[16] N. Kim, J. Cho, and E. Seo, "Energy-based accounting and scheduling of virtual machines in a cloud system," in *Proc. IEEE/ACM Int. Conf. Green Comput. Commun. (GreenCom)*, Aug. 2011, pp. 176–181.
[17] J. C. McCullough, Y. Agarwal, J. Chandrashekar, S. Kuppuswamy, A. C. Snoeren, and R. K. Gupta, "Evaluating the effectiveness of model-based power characterization," in *Proc. USENIX Annu. Tech. Conf.*, vol. 20. 2011, p. 1–14.
[18] H. Yang, Q. Zhao, Z. Luan, and D. Qian, "iMeter: An integrated VM power model based on performance profiling," *Future Generat. Comput. Syst.*, vol. 36, pp. 267–286, Jul. 2014.
[19] Q. Chen, P. Grosso, K. van der Veldt, C. De Laat, R. Hofman, and H. Bal, "Profiling energy consumption of VMs for green cloud computing," in *Proc. IEEE 9th Int. Conf. Dependable, Auto. Secure Comput. (DASC)*, Dec. 2011, pp. 768–775.
[20] C. Gu, H. Huang, and X. Jia, "Power metering for virtual machine in cloud computing-challenges and opportunities," in *Proc. IEEE Access*, vol. 2. Sep. 2014, pp. 1106–1116.
[21] R. Bertran *et al.*, "Accurate energy accounting for shared virtualized environments using PMC-based power modeling techniques," in *Proc. 11th IEEE/ACM Int. Conf. Grid Comput. (GRID)*, Oct. 2010, pp. 1–8.
[22] R. Bertran *et al.*, "Energy accounting for shared virtualized environments under DVFS using PMC-based power models," *Future Generat. Comput. Syst.*, vol. 28, no. 2, pp. 457–468, Feb. 2012.
[23] A. E. H. Bohra and V. Chaudhary, "VMeter: Power modelling for virtualized clouds," in *Proc. IEEE Int. Symp. Parallel Distrib. Process., Workshops Phd Forum (IPDPSW)*, Apr. 2010, pp. 1–8.
[24] D. Versick, I. Waßmann, and D. Tavangarian, "Power consumption estimation of CPU and peripheral components in virtual machines," *ACM SIGAPP Appl. Comput. Rev.*, vol. 13, no. 3, pp. 17–25, Sep. 2013.
[25] I. Waßmann, D. Versick, and D. Tavangarian, "Energy consumption estimation of virtual machines," in *Proc. 28th Annu. ACM Symp. Appl. Comput.*, 2013, pp. 1151–1156.
[26] P. Xiao, Z. Hu, D. Liu, G. Yan, and X. Qu, "Virtual machine power measuring technique with bounded error in cloud environments," *J. Netw. Comput. Appl.*, vol. 36, no. 2, pp. 818–828, Mar. 2013.
[27] X. Peng and Z. Sai, "A low-cost power measuring technique for virtual machine in cloud environments," *Int. J. Grid Distrib. Comput.*, vol. 6, no. 3, pp. 69–80, 2013.
[28] Z. Jiang, C. Lu, Y. Cai, Z. Jiang, and C. Ma, "VPower: Metering power consumption of VM," in *Proc. 4th IEEE Int. Conf. Softw. Eng. Service Sci. (ICSESS)*, May 2013, pp. 483–486.
[29] H. Liu, H. Jin, C.-Z. Xu, and X. Liao, "Performance and energy modeling for live migration of virtual machines," *Cluster Comput.*, vol. 16, no. 2, pp. 249–264, Jun. 2013.
[30] C. Ma *et al.*, "Virtual machine power metering and its applications," in *Proc. IEEE Global High Tech Congr. Electron. (GHTCE)*, Nov. 2013, pp. 153–156.
[31] F. Quesnel, H. K. Mehta, and J.-M. Menaud, "Estimating the power consumption of an idle virtual machine," in *Proc. IEEE Green Comput. Commun. (GreenCom)*, Aug. 2013, pp. 268–275.
[32] W.-Y. Loh, "Classification and regression trees," *Wiley Interdiscipl. Rev., Data Mining Knowl. Discovery*, vol. 1, no. 1, pp. 14–23, Jan./Feb. 2011.
[33] Y. Li, Y. Wang, B. Yin, and L. Guan, "An online power metering model for cloud environment," in *Proc. 11th IEEE Int. Symp. Netw. Comput. Appl. (NCA)*, Aug. 2012, pp. 175–180.

CHONGLIN GU received the B.S. degree in computer science from Harbin Engineering University, in 2008, and the M.S. degree in computer science from the Harbin Institute of Technology, China, in 2011. He is currently pursuing the Ph.D. degree with the School of Computer Science and Technology, Harbin Institute of Technology Shenzhen Graduate School. His research interests are in the fields of cloud computing, distributed computing, and big data, especially algorithms design and system implementation.

PENGZHOU SHI received the B.S. degree in computer science from HEU, in 2012. He is currently pursuing the degree with the School of Computer Science and Technology, Harbin Institute of Technology Shenzhen Graduate School. His research interests are in the fields of virtualization technology and security verification.

SHUAI SHI received the B.S. degree in computer science from STDU, in 2012. He is currently pursuing the degree with the School of Computer Science and Technology, Harbin Institute of Technology Shenzhen Graduate School. His research interests are in the fields of cloud computing and big data.

HEJIAO HUANG received the Ph.D. degree in computer science from the City University of Hong Kong, in 2004. She is currently a Professor with the Harbin Institute of Technology Shenzhen Graduate School, China. Her research interests include cloud computing, trustworthy computing, formal methods for system design, and wireless networks.



XIAOHUA JIA (F'13) received the B.Sc. and M.Eng. degrees from the University of Science and Technology of China, Hefei, China, in 1984 and 1987, respectively, and the D.Sc. degree in information science from the University of Tokyo, Tokyo, Japan, in 1991. He is currently a Professor with the Department of Computer Science, Harbin Institute of Technology Shenzhen Graduate School, Shenzhen, China. His research interests include cloud computing and distributed systems, computer networks, wireless sensor networks, and mobile wireless networks. He is a fellow of the IEEE Computer Society.