Wen Y, Wang WP, Guo L *et al.* Automated power control for virtualized infrastructures. JOURNAL OF COMPUTER SCIENCE AND TECHNOLOGY 29(6): 1111–1122 Nov. 2014. DOI 10.1007/s11390-014-1494-x

Automated Power Control for Virtualized Infrastructures

Yu Wen (文　雨), Wei-Ping Wang (王伟平), *Member*, *CCF*, Li Guo (郭　莉), *Senior Member*, *CCF* and Dan Meng (孟　丹), *Senior Member*, *CCF*, *Member*, *IEEE*

Institute of Information Engineering, Chinese Academy of Sciences, Beijing 100093, China

E-mail: {wenyu, wangweiping, guoli, mengdan}@iie.ac.cn

Received August 28, 2013; revised May 26, 2014.

Abstract Power control for virtualized environments has gained much attention recently. One of the major challenges is keeping underlying infrastructure in reasonably low power states and achieving service-level objectives (SLOs) of upper applications as well. Existing solutions, however, cannot effectively tackle this problem for virtualized environments. In this paper, we propose an automated power control solution for such scenarios in hope of making some progress. The major advantage of our solution is being able to precisely control the CPU frequency levels of a physical environment and the CPU power allocations among virtual machines with respect to the SLOs of multiple applications. Based on control theory and online model estimation, our solution can adapt to the variations of application power demands. Additionally, our solution can simultaneously manage the CPU power control for all virtual machines according to their dependencies at either the application-level or the infrastructure-level. The experimental evaluation demonstrates that our solution outperforms three state-of-the-art methods in terms of achieving the application SLOs with low infrastructure power consumption.

Keywords power control, virtualized infrastructure, multi-tier application, virtual machine, service-level objective

1 Introduction

The motivation of this paper stems from two growingly important trends on modern datacenters. One trend is the popularity of virtualization technology which is leading to a disruptive change on the existing resource sharing paradigms for datacenters. For instance, in Fig.1, three multi-tier applications share three virtualized nodes of a cluster, where 1) each VM hosts a function tier of an application, such as VM 1 hosting the Web tier of application A and VM 5 hosting the database tier of application B; 2) the applications may span multiple nodes, such as application A on nodes 1, 2 and 3; and 3) the VMs from the different applications may share one node, such as VM 2 of application A and VM 5 of application B share node 2. Another trend is the power management for datacenters. This problem has regained considerable attention in system design and management because of the power delivery limits, which result from the continuously increasing hardware density and the huge energy cost of the servers and the cooling system in a datacenter. However, the power management for such virtualized environments faces several challenges.

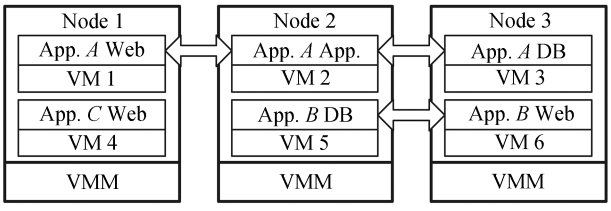


Fig.1. Virtualized cloud platform that hosts three multiple-tier applications. App.: Application. DB: Database.

The requirement to simultaneously guarantee service-level objectives (SLOs) of multiple applications, e.g., application end-to-end performance, is the first key challenge of the power management for a virtualized environment. In cloud computing, infrastructure service providers need to meet the SLOs of hosted applications in the infrastructure-as-a-service paradigm. However, most of the recently proposed solutions, e.g., [1-4], treat the power consumption as the first-class control target without considering the SLOs of application services. For example, the power management system presented in [1] uses a control theory based method to adaptively keep the power consumption of a virtualized platform below a budget. While this solution works effectively

Regular Paper

This work was supported by the National Key Technology Research and Development Program of the Ministry of Science and Technology of China under Grant No. 2012BAH46B03, the National HeGaoJi Key Project under Grant No. 2013ZX01039-002-001-001, and the Strategic Priority Research Program of the Chinese Academy of Sciences under Grant No. XDA06030200.

in controlling the power consumption of the system, it cannot strictly guarantee the application performance even with a sufficient power budget. Moreover, it is non-trivial to simultaneously achieve the SLOs of multiple applications. Thus, application SLOs oriented solutions need to be proposed for virtualized environments.

Adaptability to the time-varying application behavior is the second challenge of the power management for a virtualized environment. Usually, the application behavior, e.g., the power demand, is workload-dependent and susceptible to the environmental variations, such as those resulting from the resource contention among multiple applications in a shared environment. The variations of application workload usually not only involve the changes of the workload intensity, but also involve the changes of the workload mix, e.g., the proportion of each request class. Nevertheless, recently proposed techniques[2-4] are able to adapt to the inherent power demand changes within an application instead of the external variations from application workloads and environments. As a result, these solutions may unnecessarily decrease power provisioning when an application runs in the performance-critical phases and thus lead to deficient application performance. Therefore, a power control solution must be able to handle such realistic variations of the application behavior.

Another major challenge of the power management for virtualized environments is to coordinate the power consumption control for related application VMs in the multi-tier application scenarios. Recently a few studies[1,5] present multiple-input, multiple-output (MIMO) power control solutions based on control theory for multiple applications. However, these solutions neither explicitly control the power consumption of all VMs of an application[1], nor consider the dependencies of the power consumption of related VMs as this kind of approach is oriented to the single-tier application scenario[5]. Since the unawareness of VM dependencies can exacerbate the performance imbalance among function tiers of an application, these solutions may lead to power efficiency deterioration and even undesired application performance degradation. This issue is particularly important for cloud infrastructures whose primary workloads are expected to be multi-tier applications.

In this paper, we propose a novel and highly efficient power control system that automatically manages the power provision for a virtualized infrastructure that is specially designed for multi-tier applications. Our control solution features a two-layer design. We first adopt control theory to precisely control the performance of individual applications, with theoretically guaranteed accuracy and stability, by simultaneously calculating the power demands of all VMs of an application. Then, we dynamically adjust the power provisioning for each node and determine actual power allocations among all the VMs within a node according to their power demands and actual system conditions. Therefore, our solution can guarantee the application performance while precisely limiting the underlying power consumption as much as possible.

Specifically, this paper makes the following major contributions.

- We propose a highly efficient power control solution for a virtualized infrastructure hosting multiple multi-tier applications. Our solution is able to automatically adjust the power provisioning at the node-level and the power allocations at the VM-level according to the SLOs of applications.
- We propose an adaptive approach to dynamically describe the relationship between the SLOs of a multi-tier application and the power allocations for its VMs. This approach can adapt to the time-varying application behavior which leads to the complicated changes of the power demands of the application VMs.
- We implement our control solution on a small-scale cluster of Xen-based machines and present experimental results to demonstrate that our solution achieves better power control efficiency than three of the methods proposed in the literatures. The overhead tests for the power demand calculations also demonstrate the scalability of our solution on virtualized infrastructures.

The rest of this paper is organized as follows. Section 2 describes our problem and goals in this paper. Section 3 introduces the design for our power control system. Section 4 presents our experimental testbed. Section 5 discusses the evaluation results. Section 6 presents the related work and Section 7 concludes the paper.

2 Problem Statement

In this paper, we adopt CPU as the control object of our power management system as CPU is one of the major power-consuming components in modern computer systems. Many kinds of modern CPUs provide assistant mechanisms, such as dynamic voltage and frequency scaling (DVFS), to achieve dynamic power management. Through DVFS, CPU voltage and frequency can be intentionally decreased to reduce the CPU power consumption. In this paper, we define a metric, named CPU frequency quota, to quantify the CPU power allocations for application VMs. The CPU frequency quota is defined as the ratio of the CPU frequency expected by or allocated for an application VM to the highest CPU frequency level. Here, all of CPUs within a node are always at the same CPU frequency

level. In addition, we adopt two performance metrics, average response time and throughput, for the application SLOs.

In this paper, our goal is to design a highly efficient power management system that can automatically adjust CPU frequency levels for a virtualized infrastructure and allocate CPU frequency quotas for multi-tier applications with respect to the average response time goals and the throughput goals.

3 Design

In this section, we present the design for our power management system for virtualized infrastructures.

3.1 Overview

We first give an overview of our management system. As shown in Fig.2, our system features a two-layer design: the top layer consists of a group of application power controllers that dynamically calculate the CPU frequency quota demands for all application VMs; the bottom layer comprises a group of node power controllers that determine the CPU frequency levels of all nodes and the actual CPU frequency quota allocations for all the application VMs.

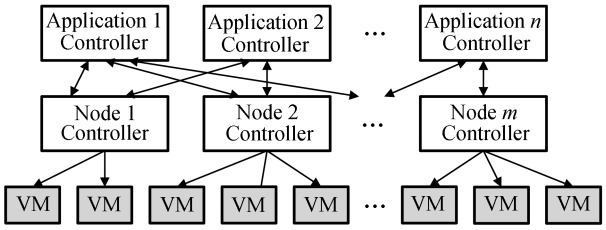


Fig.2. Structure of our power control system.

Each application power controller periodically calculates the CPU frequency quota demands for all VMs of an application and then sends power requests to those node power controllers on nodes hosting these VMs. Based on such requests from the different applications, each node power controller finally decides the CPU frequency levels for a node and the actual CPU frequency quota allocations for all the VMs within the node.

In the following subsections, we introduce the design for the application power controller and the node power controller respectively. For easy reference, Table 1 summarizes the most relevant mathematical symbols used in this paper.

3.2 Application Power Controller

Our application controller is based on control theory and consists of a basic feedback controller and an online model estimator. Fig.3 shows the control logic of the controller during each control interval k. At the start of a control interval, the feedback-based controller uses an application performance model to calculate the CPU frequency quota demands $\boldsymbol{f}(k)$ for all VMs of an application based on the difference between the performance goal $\bar{y}$ and the measured performance $y'(k-1)$ in the last control interval $k-1$ for the application. The application performance model correlates the application performance with the CPU frequency quota allocations for all the VMs of the application. At the end of the control interval, according to the measured performance $y'(k)$ and the actual CPU frequency quota allocations $\boldsymbol{f}'(k)$ for the application in the current interval, the estimator recalibrates the coefficients of the application performance model for the feedback controller to capture the newest relationship between the application performance and the CPU frequency quota allocations for all the VMs of the application.

Table 1. Notation

Mathematical Symbol	Description
$y(k)$	Predicted performance for an application in interval k
$y'(k)$	Measured performance for an application in interval k
$\bar{y}$	Performance goal for an application
$\boldsymbol{f}(k)$	Column vector of the CPU frequency quota demands for all VMs of an application in interval k
$\boldsymbol{f}'(k)$	Column vector of the actual CPU frequency quota allocation for all VMs of an application in interval k
$f_{a,i}(k)$	CPU frequency quota demand for VM i of application A in interval k
$f'_{a,i}(k)$	Actual CPU frequency quota allocation for VM i of application A in interval k
$F_j(k)$	Demanded CPU frequency for node j in interval k
$F'_j(k)$	Actual CPU frequency level for node j in interval k

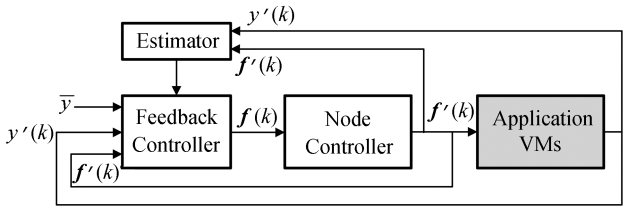


Fig.3. Control logic of our application power controller. Based on control theory, a feedback controller adaptively calculates the CPU frequency quota demands for all VMs of an application $\boldsymbol{f}(k)$ subject to the application performance goal $\bar{y}$. Based on the measured performance $y'(k)$ and the actual CPU frequency quota allocations $\boldsymbol{f}'(k)$, an estimator dynamically adjusts an application performance model for the feedback controller to adaptively capture the relationship between the application performance and its CPU frequency quota allocations.

3.2.1 Basic Feedback Controller

We use a widely adopted time series model, named auto-regressive moving average (ARMA) model, to design the application performance model for the feedback controller. Specifically, we adopt two time series of the measured performance and the actual CPU frequency quota allocations for the application as the autoregressive part and the moving average part of the ARMA-based application performance model respectively. The model can deterministically describe the locally linear relationship between the application performance and the CPU frequency quota allocations. Therefore, we can predict the future application performance with the expected CPU frequency quota allocations for the current control interval and history information which includes the measured application performance and the actual CPU frequency quota allocations in the recent control intervals. In implementation, we adopt second-level ARMA model since it has acceptable accuracy and less computational complexity. The experimental results are presented in Subsection 5.1.

Therefore, the application performance model is formulated as:

$$y(k) = a_1(k)y'(k-1) + \boldsymbol{b}_0^{\mathrm{T}}(k)\boldsymbol{f}(k) + \boldsymbol{b}_1^{\mathrm{T}}(k)\boldsymbol{f}'(k-1), \quad (1)$$

where a_1, $\boldsymbol{b}_0$ and $\boldsymbol{b}_1$ are the model coefficients that respectively correlate the predicted application performance $y(k)$ with the expected CPU frequency quota allocations $\boldsymbol{f}(k)$ for the current control interval k, the measured application performance $y'(k-1)$, and the actual CPU frequency quota allocations $\boldsymbol{f}'(k-1)$ in the last control interval $k-1$. Since $\boldsymbol{f}$ and $\boldsymbol{f}'$ are column vectors that denote the expected and the actual CPU frequency quota allocations for an application respectively, their coefficients $\boldsymbol{b}_0$ and $\boldsymbol{b}_1$ are also column vectors of the same dimension. Notice that these coefficients are also functions of the control interval k and thus can be dynamically adjusted by the online model estimator. Therefore, our solution can dynamically capture the globally non-linear relationship between the application performance and the CPU frequency quota allocations. Subsection 3.2.2 presents the design details for the online model estimator.

By now, through the model in (1), we can calculate the CPU frequency quota demands $\boldsymbol{f}(k)$ subject to the application performance goal $\bar{y}$. This kind of calculation can be regarded as a typical optimization problem. We first define our optimization objectives for this problem as:

1) minimize $|y(k) - \bar{y}|$, and
2) minimize $||\boldsymbol{f}(k)||$.

The objectives mean achieving the application performance goal with the minimal CPU power consumption.

Further, the objectives can be formulated together as a linear quadratic cost function, a common approach used for many optimization problems[6]. This cost function is:

$$J = W_y|y(k) - \bar{y}|^2 + W_{\boldsymbol{f}}||\boldsymbol{f}(k)||^2, \quad (2)$$

where W_y and $W_{\boldsymbol{f}}$ are control weights for objectives 1) and 2) respectively. Since the application performance and the CPU frequency quota allocations are two very different metrics, we use the control weights to normalize them and thus get control balance between achieving the application performance goal and minimizing the CPU power consumption. In implementation, we set W_y and $W_{\boldsymbol{f}}$ to be as $1/\bar{y}$ and $1/||\boldsymbol{f}_{\max}||$ respectively, where $||\boldsymbol{f}_{\max}||$ denotes the norm of the likely maximal CPU frequency quota allocations.

Then, we calculate the optimal CPU frequency quota allocations that minimize the cost function J. We get the first derivative of the function J on $\boldsymbol{f}(k)$. For easy description, we define $\boldsymbol{x}(k) = \boldsymbol{b}_1^{\mathrm{T}}(k)a_1(k)$, and $\boldsymbol{\varphi}(k-1) = \big((\boldsymbol{f}'(k-1))^{\mathrm{T}}y'(k-1)\big)^{\mathrm{T}}$. As $y(k)$ is replaced with (1), (2) can be rewritten as:

$$\begin{aligned} J = {} & W_y\Big(\boldsymbol{x}(k)\boldsymbol{\varphi}(k-1) - \bar{y}\Big)^2 + \\ & W_y\Big(\boldsymbol{b}_0(k)\boldsymbol{f}(k)\Big)^2 + \\ & 2W_y\boldsymbol{f}^{\mathrm{T}}(k)\boldsymbol{b}_0^{\mathrm{T}}(k)\Big(\boldsymbol{x}(k)\boldsymbol{\varphi}(k-1) - \bar{y}\Big) + \\ & W_{\boldsymbol{f}}||\boldsymbol{f}(k)||^2. \end{aligned}$$

Therefore, the first derivative of the function J on $\boldsymbol{f}(k)$ is:

$$\begin{aligned} \frac{\partial J}{\partial \boldsymbol{f}(k)} = {} & 2W_y\boldsymbol{b}_0^{\mathrm{T}}(k)\boldsymbol{b}_0(k)\boldsymbol{f}(k) + \\ & 2W_y\boldsymbol{b}_0^{\mathrm{T}}(k)\Big(\boldsymbol{x}(k)\boldsymbol{\varphi}(k-1) - \bar{y}\Big) + \\ & 2W_{\boldsymbol{f}}\boldsymbol{f}(k). \end{aligned}$$

Notice that the second derivative of the function J on $\boldsymbol{f}(k)$ is a positive constant $2W_y\boldsymbol{b}_0^{\mathrm{T}}(k)\boldsymbol{b}_0(k) + 2W_{\boldsymbol{f}}$, thus the minimum of the function J always exists.

Finally, as the first derivative is equal to zero, we can get the optimal CPU frequency quota allocations:

$$\begin{aligned} \boldsymbol{f}(k) = {} & \Big(W_y\boldsymbol{b}_0^{\mathrm{T}}(k)\boldsymbol{b}_0(k) + W_{\boldsymbol{f}}\boldsymbol{I}\Big)^{-1} \cdot \\ & W_y\boldsymbol{b}_0^{\mathrm{T}}(k)\Big(\bar{y} - \boldsymbol{x}(k)\boldsymbol{\varphi}(k-1)\Big), \end{aligned}$$

where $\boldsymbol{I}$ is an identity matrix.

3.2.2 Online Model Estimator

As presented in the previous subsection, the application performance model in (1) is designed to describe locally linear relationship between the application performance and the CPU frequency quota allocations. In this subsection, we present the online model estimator that allows our system to dynamically capture globally non-linear relationship between them.

The online estimator is designed to dynamically estimate the coefficients of the application model in (1). In classic control theory, least squares (LS) is a main stream approach for model estimation. Here, we adopt recursive least squares (RLS)[7], an online variant of ordinary LS, to implement our dynamic model estimation. RLS not only possesses the same accuracy as ordinary LS, but also has less computational complexity because of its recursive feature.

For easy description, we rewrite (1) as $y(k) = \boldsymbol{z}(k)\boldsymbol{\theta}(k)$, where we define $\boldsymbol{z}(k) = \boldsymbol{b}_0^{\mathrm{T}}(k)\boldsymbol{b}_1^{\mathrm{T}}(k)a_1(k)$ and $\boldsymbol{\theta}(k) = \big((\boldsymbol{f}'(k))^{\mathrm{T}}(\boldsymbol{f}'(k-1))^{\mathrm{T}}y'(k-1)\big)^{\mathrm{T}}$. Therefore, the RLS-based coefficient estimation for the mode in (1) can be described as:

$$\varepsilon(k) = y'(k) - \hat{\boldsymbol{z}}(k)\boldsymbol{\theta}(k), \tag{3}$$

$$\hat{\boldsymbol{z}}(k) = \hat{\boldsymbol{z}}(k-1) + \frac{\varepsilon(k)\boldsymbol{\theta}^{\mathrm{T}}(k)\boldsymbol{A}(k-2)}{1+\boldsymbol{\theta}^{\mathrm{T}}(k)\boldsymbol{A}(k-2)\boldsymbol{\theta}(k)}, \tag{4}$$

$$\boldsymbol{A}(k-1) = \boldsymbol{A}(k-2) + \frac{\boldsymbol{A}(k-2)\boldsymbol{\theta}(k)\boldsymbol{\theta}^{\mathrm{T}}(k)\boldsymbol{A}(k-2)}{1+\boldsymbol{\theta}^{\mathrm{T}}(k)\boldsymbol{A}(k-2)\boldsymbol{\theta}(k)}, \tag{5}$$

where $\varepsilon(k)$ is the error of the application performance prediction, $\hat{\boldsymbol{z}}(k)$ indicates an estimated value for $\boldsymbol{z}(k)$ and therefore $\hat{\boldsymbol{z}}(k)\boldsymbol{\theta}(k)$ is the expected application performance for the actual CPU frequency quota allocations $\boldsymbol{f}'(k)$ for the current control interval, and $\boldsymbol{A}(k)$ is a covariance matrix. At the end of each control interval, if the prediction error $\varepsilon(k)$ that is calculated via (3) is not negligible, the estimator will re-calculate the coefficients of (1) following (4) and (5).

3.3 Node Power Controller

After calculating the CPU frequency quota allocations demands for an application, each application power controller sends power requests to the node power controllers on those nodes hosting the VMs for the application. According to such requests from the different application controllers, each node power controller determines the CPU frequency levels for its node and the actual CPU frequency quota allocations for all the application VMs within the node.

In this subsection, we present our approach for the CPU frequency level adjusting and the CPU frequency quota allocations following the example shown in Fig.1. In particular, we discuss the power control for VM 3 and VM 6, respectively, belonging to applications A and B, respectively, on node 3. We suppose that the CPU frequency quota demands calculated for these two VMs in control interval k are $f_{a,3}(k)$ and $f_{b,6}(k)$ respectively. As presented in Section 2, we define the CPU frequency quota as the ratio of the CPU frequency expected by a VM to the highest CPU frequency level. Therefore, the expected CPU frequency level for node 3 $F_3(k)$ is equal to $(f_{a,3}(k) + f_{b,6}(k))F^*$, where F^* denotes the highest CPU frequency level. Since the feasible CPU frequency levels are a set of discrete values, we need to find an appropriate level $F_3'(k)$ that is either immediately higher than $F_3(k)$ or the highest level if $F_3(k)$ exceeds the range of the feasible levels. Finally, the actual CPU frequency quota allocations $f'_{a,3}(k)$ and $f'_{b,6}(k)$, for VM 3 and VM 6, are $f_{a,3}(k)F_3'(k)/(f_{a,3}(k)F^* + f_{b,6}(k)F^*)$ and $f_{b,6}F_3'(k)/(f_{a,3}(k)F^* + f_{b,6}(k)F^*)$ respectively.

In implementation, we use Xen's Xenpm utility to set the CPU frequency levels. Xenpm provides an interface that allows users to specify the expected CPU frequency levels. Furthermore, we use Xen's Credit scheduler to enforce the CPU frequency quota allocations for the application VMs. We set each VM's CPU cap equal to its CPU frequency quota. The CPU cap denotes the maximum amount of the CPU resources that a VM can consume during each unit time. Then, the Credit scheduler allocates the CPU resources for the VMs according to their CPU caps.

4 Experimental Testbed

We evaluate our solution on a Xen-based cluster where each machine has two 8-core Intel Xeon E5-2670 processors, 28 G memory, 146 GB disk and a 1 Gbps NIC, and runs OpenSuSE 10.3 and Xen 3.4.1 (for Linux 2.6.22.5-31 SMP kernel). The processor has 15 possible frequency levels, ranging from 1.2 GHz to 2.6 GHz. All the machines are connected via an adaptive 1 Gbps switcher within the cluster.

Moreover, we adopt two test applications: an RUBiS system① and the TPC-W benchmark②. RUBiS system implements an online bid service. It has a three-tier structure that consists of a web server, an application logic tier and a MySQL database. RUBiS system supports 22 kinds of requests and two kinds of workload mixes: browsing and bidding. The TPC-W benchmark is also a multi-tier service and simulates an online bookstore service. It supports 14 kinds of requests and three

①http://rubis.orjectweb.org, Sept. 2014.

②http://www.tpc.org/tpcw, Sept. 2014.

kinds of workload mixes: browsing, shopping, and ordering.

We design experimental workloads for the applications with a part of access logs from the official World Cup website in 1998[8]. Fig.4(a) shows a statistic curve of average request arrival rate for the trace. Based on these statistics, we synthesize five kinds of application workloads shown in Fig.4(b). Since our experiments are to test the adaptability of our solution, we simulate the workload variations instead of the real request arrival rates. Specifically, we vary the number of five kinds of application clients over time according to the statistics shown in Fig.4(b). The five kinds of clients implement RuBiS' browsing and bidding workload, and TPC-W's browsing, shopping, and ordering workload respectively. In addition, in order to simulate the power demands shifting between the applications, we alternately generate the workload variations for the applications. As presented in Section 2, we adopt average response time and throughput as the metrics for the application SLOs.

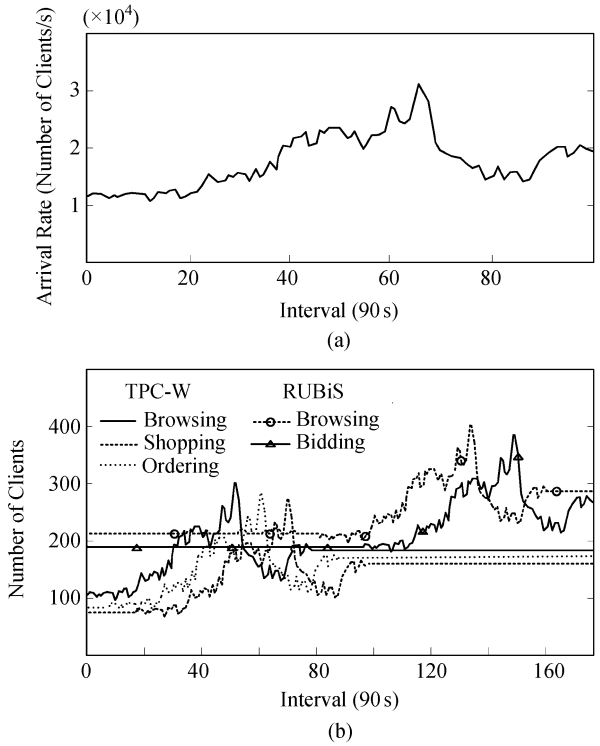


Fig.4. Experimental workload synthesized using a part of a real trace of the World Cup 1998 website. (a) A part of the trace of the World Cup 98 website. (b) Simulation workloads.

Additionally, we compare the experimental results for our solution with those for the three aforementioned state-of-the-art methods. The first method that we denote as *Prediction*, is a DVFS policy based on application workload prediction proposed by Rong *et al.*[9] This method adopts average request arrival rate as an indicator for the workload intensiveness and predicts such metric with its past information. In addition, this method requires to specify a bound on the performance degradation when adjusting the CPU frequency levels. For fairness, we use the average performance degradation from our solution as the bound for this method. The second method that we denote as *Utility*, is a heuristic solution like the Xenpm's ondemand mechanism that dynamically adjusts the CPU frequency levels based on CPU utilization observation. The third method that we denote as *Offline*, in fact, is an offline version of our solution. This method supposes that the relationship between the application performance and the power allocations is deterministic and can be identified using samples of the application workload. Therefore, unlike our solution, its application performance model is built in an offline manner that the model coefficients are estimated statically. In particular, we use the workload shown in Fig.4(b) and a set of, intentionally generated, control inputs to test the system and then collect the control outputs. Then, we use ordinary Least Squares method on these data to estimate the model coefficients. Once the model is determined, it cannot be adjusted during the experiments. Correspondingly, we refer to our solution as *AutoFreq* in our experiments.

In our experiments, we first evaluate ARMA model accuracy with respect to the model levels. Second, we compare the experimental results in terms of the application performance control and the power consumption control for *AutoFreq*, *Prediction*, *Utility*, and *Offline* on four of the nodes of the cluster respectively. The experimental setup is shown in Fig.5. Three nodes work together as a shared platform for the RUBiS system and the TPC-W benchmark. Another node hosts two application power controllers for the applications. Application clients run on the rest of the machines. At

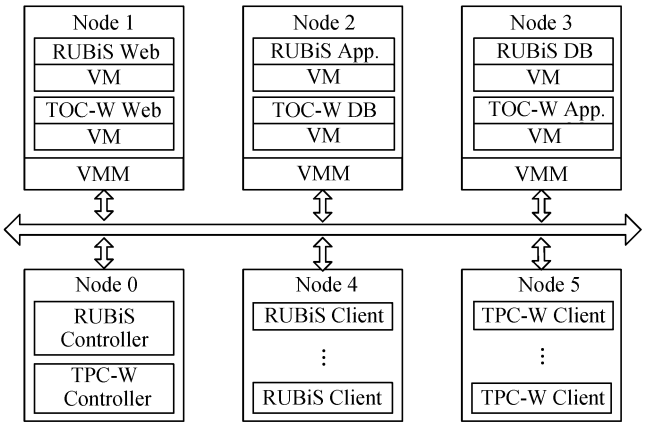


Fig.5. Experimental setup for comparing our system with the three proposed methods. App.: Application, DB: Database.

last, we evaluate the scalability of *AutoFreq* with up to 384 VMs on different nodes of the cluster.

5 Evaluation Results and Discussions

5.1 ARMA Model Accuracy

As presented in Subsection 3.2.1, our application performance model is based on the ARMA model. An important issue to the ARMA model applications is to determine the model levels. While a higher model level is able to bring ARMA model higher accuracy, the complexity of the computation based on the model inevitably increases. In this subsection, in order to find a cost-efficient ARMA model, we compare the accuracy of four ARMA models with four model levels respectively. We use the error of the application performance prediction to quantify the model accuracy. Furthermore, we adopt three CPU frequency levels: 1.2 GHz, 1.9 GHz and 2.6 GHz. For evaluation accuracy, tests of each experimental configuration are repeated 20 times. In each test, the nodes dedicatedly host a single application and all CPUs are at the same frequency level.

In Fig.6, the experimental results for the ARMA model accuracy evaluation are shown. Obviously, the first-level model has higher prediction errors (26.1% in average) than the other three models (11.7% at most). Without history information in the stochastic impulse, the first-level model cannot generate complete time series for the application performance. Although the models with the higher levels are more accurate than the second-level model in Fig.6, their differences are not significant (3.2% at most). Since the model level is normally a compromise between the model accuracy and the model simplicity, we finally adopt second-level ARMA model in this paper.

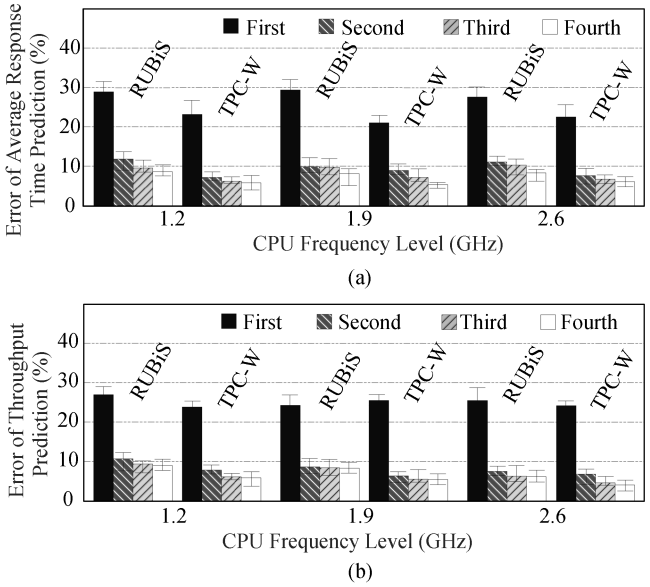


Fig.6. Comparison of the average prediction errors in four ARMA-based application performance models with different model levels. Considering a compromise between the model accuracy and the model simplicity, we choose the second-level model.

5.2 Application End-to-End Performance

In this subsection, we compare the experimental results of the application performance control for *AutoFreq*, *Prediction*, *Utility* and *Offline*. Since our goal is to minimize the power consumption while guaranteeing the application performance, comparing the original metrics for the application performance is not helpful. Thus we define two metrics for the evaluation for the application performance control. We define SLO success ratio, which is proportional to the total control intervals in which the application performance is achieved, and SLO deviation ratio, which is the ratio of the absolute performance error to the application performance goal. Here, we do not discriminate the positive errors from the negative errors since the positive ones mean unnecessary power consumption that should be avoided.

In Fig.7, the CDFs of the per-interval application performance for *AutoFreq*, *Prediction*, *Utility* and *Offline* are shown respectively. The success ratios of *AutoFreq* with respect to the average response time goal and the throughput goal for RUBiS are 80.2% and 71.4% respectively. The corresponding results for TPC-W are 77.5% and 69.9%. They are higher than the corresponding results of *Prediction* (68.2%, 58%, 65.8% and 53.4%), *Utility* (73.8%, 66.1%, 72.6% and 59.3%), and *Offline* (42.2%, 46.8%, 46.5% and 38.3%). Thus, *AutoFreq* is better than the three proposed methods with regard to the success ratio of the application performance control.

Sometimes the success ratios of *Prediction* and *Utility* are relatively comparable to those of *AutoFreq*. For example, in Fig.7(b), the success ratio of *Utility* (66.1%) differs only by a 5.3% from that of *AutoFreq* (71.4%). However, *AutoFreq* is obviously better than the two methods with respect to the average deviation ratio. In Fig.7(b), the average deviation ratios of *Prediction* and *Utility* (31.7% and 16.8% respectively) are about four times and twice higher than that of *AutoFreq* (7.3%), respectively. Thus, *AutoFreq* completely outperforms the three proposed methods with respect to achieving the application SLOs.

5.3 Power Consumption

In our experiments, we use a WattsUp power meter[③] to measure the CPU power consumption of the appli-

[③]https://www.wattsupmeters.com/secure/index.php, Sept. 2014.

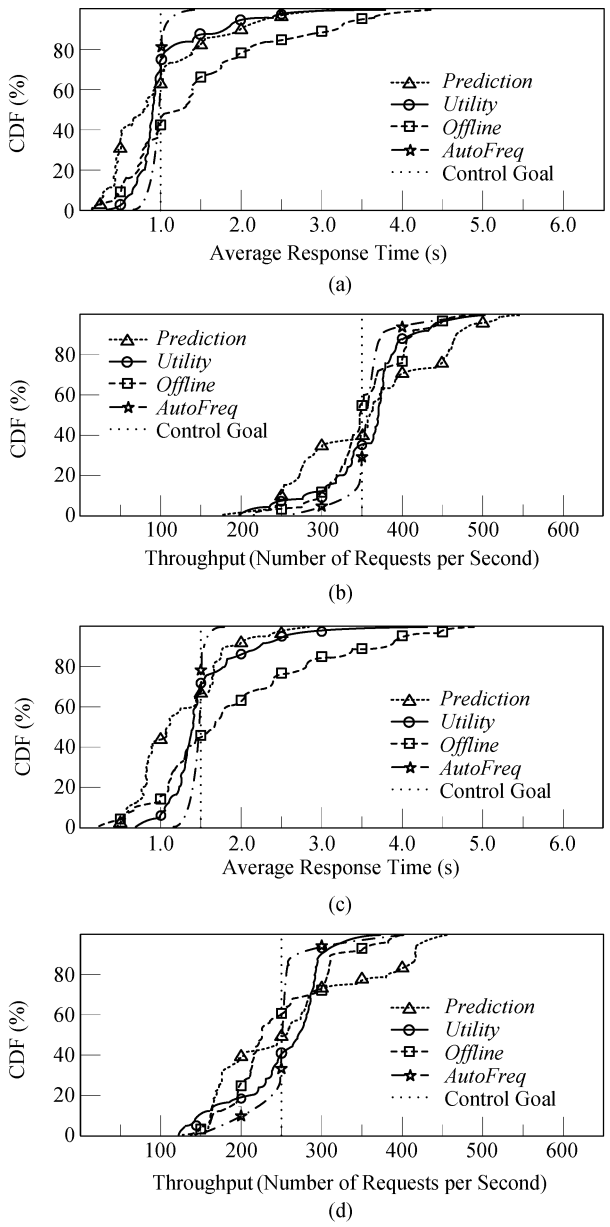


Fig.7. CDFs of the application performance for *AutoFreq*, *Prediction*, *Utility*, and *Offline* respectively. (a) CDFs of the average response time of RUBiS. (b) CDFs of the throughput of RUBiS. (c) CDFs of the average response time of TPC-W. (d) CDFs of the throughput of TPC-W.

cations. For accuracy, tests of each experimental configuration are repeated 20 times. Fig.8 shows the experimental results of the average application power consumption for *AutoFreq*, *Prediction*, *Utility* and *Offline* respectively. The results for *AutoFreq* with regard to the average response time goals and the throughput goals for the applications are 819 watts and 765 watts respectively. They are lower than the corresponding results for *Prediction* (1 115 watts and 1 021 watts) and *Utility* (994 watts and 910 watts). Therefore, *AutoFreq* outperforms *Prediction* and *Utility* in terms of the application power consumption control.

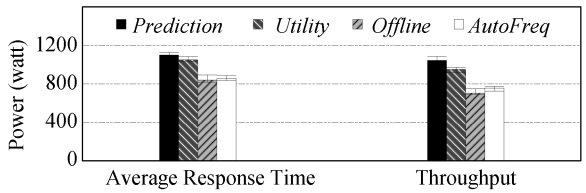


Fig.8. Comparison of the CPU power consumption for *AutoFreq*, *Prediction*, *Utility* and *Offline*. *AutoFreq* outperforms *Prediction* and *Utility* in terms of power consumption control. Compared to *Offline*, *AutoFreq* has significantly higher application performance shown in Fig.7 with little extra cost of power consumption. *AutoFreq* is better than *Offline* in terms of power efficiency.

Although *Offline* slightly outperforms *AutoFreq* in terms of power consumption control in Fig.8, *AutoFreq* significantly benefits the application performance presented in Subsection 5.2 with little extra cost of power consumption (5.2% in average). The reason is that the workloads shown in Fig.4(b) that are used in our experiments have also been used to estimate the control models of *Offline* presented in Section 4. Therefore, *Offline* can relatively easily figure out the total power demands of applications during the experiments and hence effectively manage the global power control. Nevertheless, as presented in Section 4, this method adopts offline model estimation, thus it cannot accurately capture the local variations of the application power demands. This is why *Offline* is noticeably worse than *AutoFreq* in terms of the application performance control presented in Subsection 5.2. The further investigations on the differences between *AutoFreq* and the three state-of-the-art methods are presented in the following part of this subsection.

We first compare the aggregated CPU frequency level distributions with regard to the application performance goals for *AutoFreq*, *Prediction*, and *Utility*. The experimental results are shown in Fig.9. Compared to *AutoFreq*, the results for *Prediction* and *Utility* obviously focus on high CPU frequency levels (from 2.6 GHz to 2.1 GHz). This is why the power consumption for these two methods is higher than that for *AutoFreq* shown in Fig.8.

Notice that although *Prediction* and *Utility* prefer to allow the applications to consume more CPU power, *AutoFreq*, on the contrary, outperforms them in terms of application performance control as shown in Fig.7. We further compare the traces of the aggregated CPU frequency demands calculated for the applications for these three methods. The experimental results are

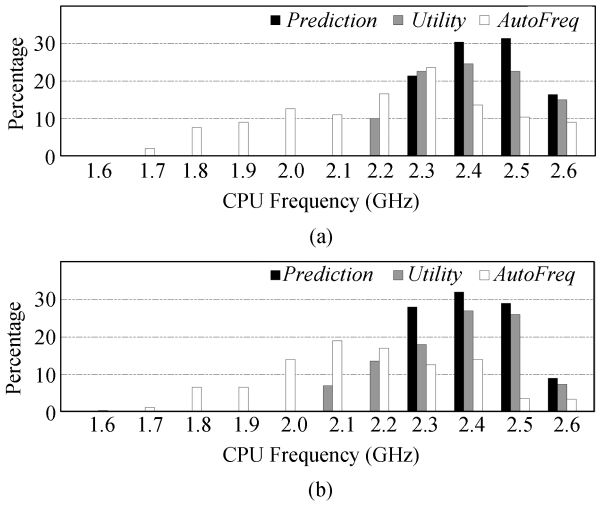


Fig.9. Comparison of the aggregated CPU frequency level distributions for *AutoFreq*, *Prediction*, and *Utility*. *AutoFreq* prefers to lower the CPU frequency levels. (a) Aggregated CPU frequency level distributions with regard to the average response time goals for the applications. (b) Aggregated CPU frequency level distributions with regard to the throughput goals for the applications.

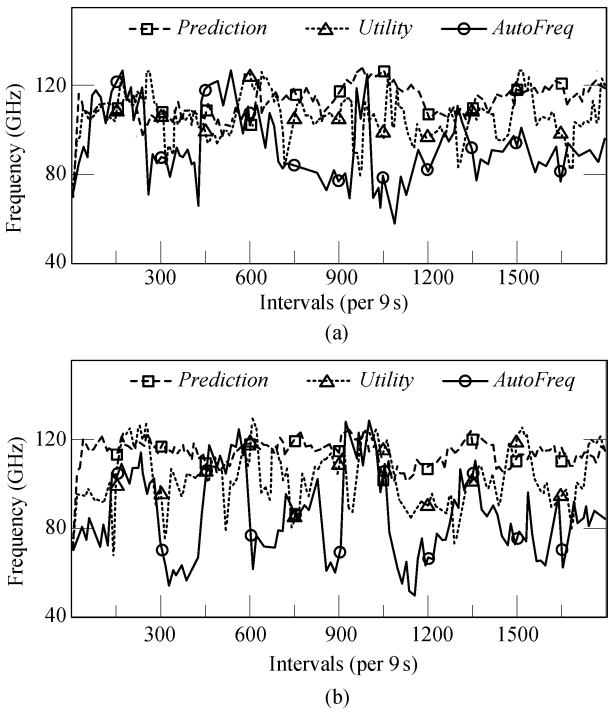


Fig.10. Comparison of the traces of the aggregated CPU frequency demands calculated for the applications for *AutoFreq*, *Prediction*, and *Utility*. *AutoFreq* is more sensitive to the variations of the application power demands than *Prediction* and *Utility*. (a) Aggregate CPU frequency demands with regard to the average response time goals for the applications. (b) Aggregate CPU frequency demands with regard to the throughput goals for the applications.

shown in Fig.10. The curves for *Prediction* and *Utility* are relatively more stable than those for *AutoFreq*. *Prediction* calculates the CPU power demands for the applications according to the predictions for the request arrival rate. However, the variations of the application power demands not only result from the changes of the workload volume, but also heavily depend on the changes of the workload mix. Thus, *Prediction* cannot effectively identify the exact CPU power demands of the applications.

Since *Utility* calculates the application power demands based on separated observations on the CPU utilization, its difficulty is the lack of the awareness of VM dependencies. Thus, this method likely leads to the performance imbalance among the VMs and thus unnecessary CPU power consumption. Based on control theory, *AutoFreq* directly correlates the application performance with the CPU power allocations instead of depending on any system factor to instruct the power control because of the complexities of the application power demand variations. Therefore, *AutoFreq* outperforms these two proposed methods in terms of achieving the application performance goals while minimizing the power consumption.

Moreover, we compare the aggregated CPU frequency level distributions and the aggregated CPU frequency quota allocations for the application for *AutoFreq* and *Offline*. For easy comparison, the results for *AutoFreq* are normalized to those for *Offline*. In addition, the CPU frequency quota allocations are aggregated at the different CPU frequency levels. The experimental results with respect to the application performance goals are shown in Figs. 11 and 12 respectively.

Compared to the differences between the aggregated CPU frequency level distributions of these two methods in Fig.11(a) and Fig.12(a), their differences of the aggregated CPU frequency quota allocations for the applications change more notably in Figs. 11(b) and 11(c) and Figs. 12(b) and 12(c). The average differences between the aggregated CPU frequency level distributions of *AutoFreq* and *Offline* with regard to the average response time goals and the throughput goals are 18.5% and 24.6% respectively. Correspondingly, the average differences between their aggregated CPU frequency quota allocations with regard to the application goals are 53.2% and 47.1% respectively.

In particular, the difference between the aggregated distributions of the 2.0 GHz frequency level of *AutoFreq* and *Offline* is relatively trivial (17%) in Fig.11(a). However, the difference between their aggregated CPU frequency quota allocations for TPC-W at the same CPU frequency level is significant (224%) in Fig.11(c). Since *Offline* adopts offline model estimation, it cannot

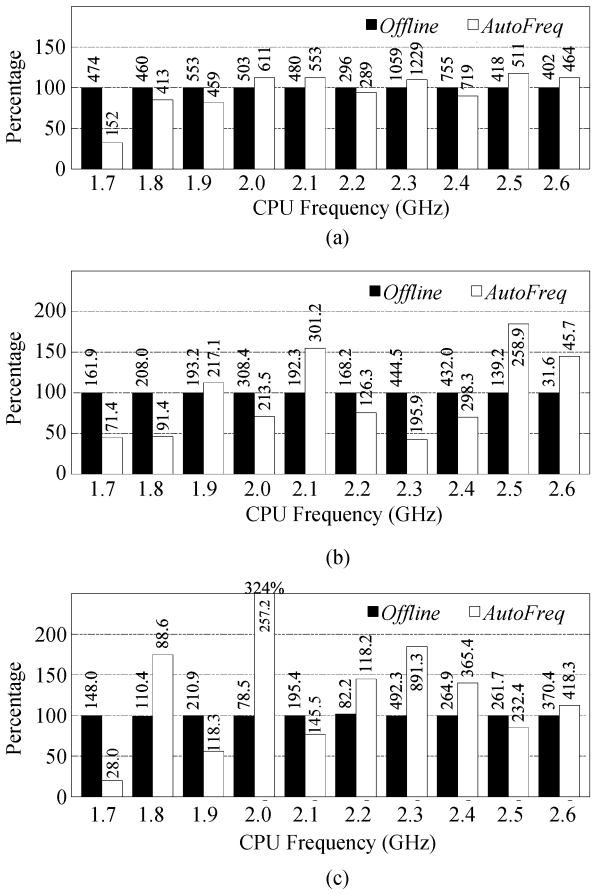


Fig.11. Comparison of the aggregated CPU frequency level distributions and the aggregated CPU frequency quota allocations for the applications for *AutoFreq* and *Offline* with respect to the average response time goals. (a) Normalized aggregated CPU frequency distributions. (b) Normalized aggregated CPU frequency quota allocations for RUBiS. (c) Normalized aggregated CPU frequency quota allocations for TPC-W.

effectively identify the local variations of the application power demands. Therefore, while the measured application power consumption for *AutoFreq* and *Offline* shown in Fig.8 is comparable, the results with regard to achieving the application goals for them presented in Subsection 5.2 are significantly different.

5.4 Scalability

In this subsection, we evaluate the scalability of *AutoFreq* with its overhead when calculating the CPU frequency quota demands for the applications. We use eight kinds of environments that include: three nodes with 3 VMs, 6 VMs, 12 VMs, 24 VMs, 48 VMs and 96 VMs respectively; six nodes with 192 VMs; and 12 nodes with 384 VMs. For accuracy, tests of each experimental configuration are repeated 20 times.

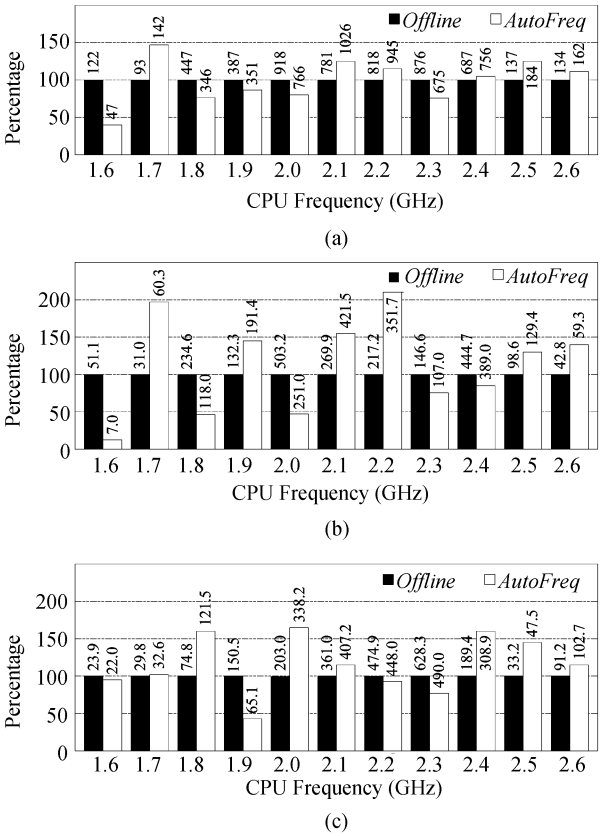


Fig.12. Comparison of the aggregated CPU frequency level distributions and the aggregated CPU frequency quota allocations for the applications for *AutoFreq* and *Offline* with respect to the throughput goals. (a) Normalized aggregated CPU frequency distributions. (b) Normalized aggregated CPU frequency quota allocations for RUBiS. (c) Normalized aggregated CPU frequency quota allocations for TPC-W.

Fig.13 shows the average calculation overhead for eight different kinds of environments. We find that the overhead does not significantly increase until it is up to 192 VMs. Moreover, compared to the control interval, the overhead, even with 384 VMs, is relatively negligible. The results imply that *AutoFreq* leads to low calculation overhead and thus features scalability.

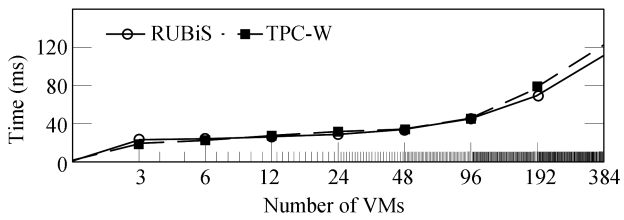


Fig.13. Overhead of the application power demand calculation with eight kinds of VM scales for *AutoFreq*. Compared to the control interval, the overhead is relatively negligible. Thus, *AutoFreq* features scalability.

6 Related Work

6.1 Power Management for Virtualized Platform

Recent work on power management for virtualized environments can be divided in two directions: control theory based methods[1,5] and non-feedback control methods[2-4,10-14]. Lim *et al.*[1] proposed a control theory based power budgeting technique for a virtualized datacenter. However, this method explicitly controls the power consumption of only one VM of a multi-tier application. In addition, this work adopts offline system identification to estimate its control model.

Wang *et al.*[5] presented a control theory based mechanism, which includes two independent control loops for the system-level power control and the application-level performance control respectively, for a virtualized cluster. However, this work first needs to carefully coordinate these two control loops to avoid control conflicts. Second, this approach aims to limit the total power consumption to be below the capacity of the power supplies instead of controlling the power consumption with respect to guaranteeing the application performance. Third, this method is designed for the scenario of single tier applications. It is unaware of the dependencies among the related VMs within a multi-tier application. Last, this work also adopts offline system identification. In contrast, our solution directly correlates with these two different control goals and thus can control the power consumption with regard to the application performance goals. Furthermore, our solution can simultaneously control the power consumption of all VMs of a multi-tier application.

Stoess *et al.*[2] and Nathuji *et al.*[12] proposed invasive power control methods for virtualized systems. This kind of techniques needs to modify the system software, e.g., OS[2] or VM hypervisor[12]. Kansal *et al.*[3] and Chen *et al.*[10] proposed non-invasive control approaches. But these methods depend on specific hardware supports, such as hardware performance counters[3] or a sensor network to monitor the system power consumption[10]. Moreover, all of these studies regard the power consumption as the first-class control target. Although some studies[4,13-14] consider the system performance, instead of being oriented to the end-to-end performance of multi-tier applications, they are proposed for low-level hardware[4], high performance computing jobs[13-14], or a single-tier web service[11]. Our solution neither modifies the source code of both the system software and an application nor depends on specific hardware support. In addition, our solution can explicitly guarantee the end-to-end performance of a multi-tier application.

6.2 Control Theory Based Methods for Other Problem

Control theory based methods are widely used for system management including performance management[15-18] and resource allocation[19-20]. Kamra *et al.*[15] proposed a request admission control mechanism to prevent a server from overload. This work adopts an adaptive PI controller to control the application performance. Padala *et al.*[19] presented a resource control method for multi-tier applications based on control theory. Lu *et al.*[20] proposed a feedback control based QoS differentiation method for a web content service. Abdelzaher *et al.*[16] presented a control theory based approach of application management for performance isolation and QoS differentiation. Karlsson *et al.*[17] designed a performance isolation mechanism for a storage system based on control theory. Wang *et al.*[18] presented a feedback control based QoS management method for an application. Except for [15, 17-18], these studies adopt offline system identification to build static control models. However, online estimation-based work proposed in [15, 17-18] is not oriented to the power control for a virtualized environment.

7 Conclusions

Power management for virtualized environments has gained much attention recently resulting from two trends: the popularity of virtualization technology and the urgencies of power management because of the continuously increasing density of hardware and the huge energy cost of modern datacenters. In this paper, we proposed an automated power control system with regard to achieving the application SLOs for virtualized infrastructures. Based on classic control theory and online application model estimation, our solution adapts to the time-varying application power demands. Additionally, our solution can simultaneously control the power consumption of all VMs of an application. Our experimental evaluation demonstrates that our solution outperforms three state-of-the-art methods in terms of achieving the application SLOs while minimizing the power consumption.

Acknowledgement We thank Lei Wang for his insightful comments. We also thank Yuan-Sheng Chen for his help on experiments. We would thank the anonymous reviewers for their constructive suggestions. Finally, we thank the editors for their helpful work.

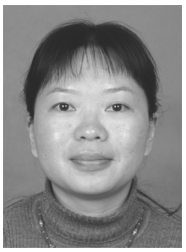
References

[1] Lim H, Kansal A, Liu J. Power budgeting for virtualized data centers. In *Proc. 2011 USENIX Annual Technical Conference*, June 2011, pp.59-72.

[2] Stoess J, Lang C, Bellosa F. Energy management for hypervisor-based virtual machines. In *Proc. 2007 USENIX Annual Technical Conference*, June 2007, pp.1-14.
[3] Kansal A, Zhao F, Liu J, Kothari N, Bhattacharya A A. Virtual machine power metering and provisioning. In *Proc. the 1st ACM Symp. Cloud Computing*, June 2010, pp.39-50.
[4] Dhiman G, Marchetti G, Rosing T. vGreen: A system for energy-efficient management of virtual machines. *ACM Trans. Design Automation of Electronic Systems*, 2010, 16(1): Article No. 6.
[5] Wang X, Wang Y. Coordinating power control and performance management for virtualized server clusters. *IEEE Trans. Parallel and Distributed Systems*, 2010, 22(2): 245-259
[6] Anderson B D O, Moore J B. Optimal Control: Linear Quadratic Methods. Prentice Hall, 1989.
[7] Astrom K J, Wittenmark B. Adaptive Control. Addison-Wesley, 1995.
[8] Arlitt M, Jin T. Workload characterization of the 1998 World Cup Web site. Technical Report, HPL-1999-35R1, HP Laboratories, Sept. 1999. http://www.hpl.hp.com/techreports/1999/HPL-1999-35R1.html, Sept. 2014.
[9] Ge R, Feng X, Feng W C, Cameron K W. CPU miser: A performance-directed, run-time system for power-aware clusters. In *Proc. the 2007 International Conference on Parallel Processing* (*ICPP*), Sept. 2007, pp.18-25.
[10] Chen H, Song M, Song J, Gavrilovska A, Schwan K. HEaRS: A hierarchical energy-aware resource scheduler for virtualized data centers. In *Proc. IEEE International Conference on Cluster Computing*. Sept. 2011, pp.508-512.
[11] Petrucci V, Carrera E V, Loques O, Leite J C B, Mosse D. Optimized management of power and performance for virtualized heterogeneous server clusters. In *Proc. the 11th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing*, May 2011, pp.23-32.
[12] Nathuji R, Schwan K. Virtualpower: Coordinated power management in virtualized enterprise systems. In *Proc. the 21st ACM SIGOPS Symp. Operating Systems Principles* (*SOSP*), Oct. 2007, pp.265-278.
[13] Takouna I, Dawoud W, Meinel C. Energy efficient scheduling of HPC-jobs on virtualized clusters using host and VM dynamic configuration. *ACM SIGOPS Operating Systems Review*, 2012, 46(2): 19-27.
[14] Zhang Z, Guan Q, Fu S. An adaptive power management framework for autonomic resource configuration in cloud computing infrastructures. In *Proc. the 31st IEEE Int. Performance Computing and Communications Conference*, Dec. 2012, pp.51-60.
[15] Kamra A, Misra V, Nahum E M. Yaksha: A self-tuning controller for managing the performance of 3-tiered Web sites. In *Proc. the 12th IEEE Int. Workshop on Quality of Service* (*IWQoS*), June 2004, pp.47-56.
[16] Abdelzaher T F, Shin K G, Bhatti N. Performance guarantees for Web server end-systems: A control-theoretical approach. *IEEE Trans. Parallel and Distributed Systems*, 2002, 13(1): 80-96.
[17] Karlsson M, Karamanolis C T, Zhu X. Triage: Performance differentiation for storage systems using adaptive control. *ACM Trans. Storage*, 2005, 1(4): 457-480.
[18] Wang X, Jin S, Xia M. Distributed quantitative QoS control based on control theory in Web cluster. *Journal of Software*, 2007, 18(11): 2810-2818. (In Chinese)
[19] Padala P, Shin K G, Zhu X, Uysal M, Wang Z, Singhal S, Merchant A, Salem K. Adaptive control of virtualized resources in utility computing environments. In *Proc. the 2nd ACM SIGOPS/EuroSys European Conference on Computer Systems*, Mar. 2007, pp.289-302.
[20] Lu Y, Abdelzaher T F, Saxena A. Design, implementation, and evaluation of differentiated caching services. *IEEE Trans. Parallel and Distributed Systems*, 2004, 15(5): 440-452.

Yu Wen received his Ph.D. degree in computer science from Institute of Computing Technology, Chinese Academy of Sciences, Beijing, in 2011. He is now an assistant professor of Institute of Information Engineering, Chinese Academy of Sciences, Beijing. His research interests include cloud computing, data management, and system security.

Wei-Ping Wang received his Ph.D. degree in computer science from Harbin Institute of Technology in 2006. He is now a professor of Institute of Information Engineering, Chinese Academy of Sciences, Beijing. His research interests include cloud computing and big data. He is a member of CCF.



Li Guo received her M.S. degree in computer science from Institute of Computing Technology, Chinese Academy of Sciences, Beijing, in 1994. She is now a professor of Institute of Information Engineering, Chinese Academy of Sciences, Beijing. Her research interests include information security and data stream analysis. She is a senior member of CCF.

Dan Meng received his Ph.D. degree in computer science from Harbin Institute of Technology in 1995. He is now a professor of Institute of Information Engineering, Chinese Academy of Sciences, Beijing. His research interests include high performance computing and computer architecture. He is a senior member of CCF and a member of IEEE.