Autonomic Resource Management for Optimized Power and Performance in Multi-tenant Clouds

Selome Kostentinos Tesfatsion, Eddie Wadbro, and Johan Tordsson
Department of Computing Science
Umeå University, Sweden
{selome, eddiew, tordsson}@cs.umu.se

***Abstract*—We present an autonomic resource management framework that takes advantage of both virtual machine resizing (CPU and memory) and physical CPU frequency scaling to reduce the power consumption of servers while meeting performance requirements of colocated applications. We design online performance and power model estimators that capture the complex relationships between applications' performance and server power (respectively), and resource utilization. Based on these models, we devise two optimization strategies to determine the most power efficient configuration. We also show that an operator can tune the tradeoff between power and performance. Our evaluation using a set of cloud benchmarks compares the proposed solution in power savings against the Linux *ondemand* and *performance* CPU governors. The results show that our solution achieves power savings between 12% to 20% compared to the baseline performance governor, while still meeting applications' performance goals.**

I. Introduction and Motivation

Reducing power consumption is a major issue in the design and operation of large-scale data centers [1]. A recent survey reports that data centers account for between 1.1% and 1.5% of the total electricity use worldwide and this share is projected to rise even more [1]. Energy consumption has become a very important concern, which in turn, motivates in-depth investigations of techniques to improve the energy efficiency of these computing infrastructures [2].

In terms of work done per energy unit consumed, a server tends to be more energy efficient at higher utilization [3]. This observation is a major reason for workload co-location in modern data centers. Nevertheless, most cloud facilities operate at very low utilization. Delimitrou et al. [4] show that a cluster at Twitter with thousands of servers consistently achieves aggregate CPU utilization below 20% and memory usage of 45% even though reservations reach up to 80% of total capacity for both resources. Utilization estimates are even lower for facilities that do not co-locate workloads. Energy wastage at lower utilization compared to an ideal, fully utilized system is significant. To improve the cost effectiveness of these systems, it is also important to improve energy efficiency at low and moderate utilization [5].

In the research literature, a large body of work applies Dynamic Voltage and Frequency Scaling (DVFS) and Vary-On/Vary-Off (VOVO) power management mechanisms. DVFS changes the operating frequency and voltage of a given computing resource. VOVO turns servers on and off to adjust the number of active servers according to the workload [6], [7]. Although the VOVO technique is one effective method for improving energy efficiency, server consolidation may not always be possible. For example, migration of stateful services is time consuming and adds significant overhead. Delays due to migration of such services may cause Service Level Objective (SLO) violations. On the other hand, existing power management systems based on DVFS, including the Linux ondemand CPU governor [8] and most OS implementations, change CPU frequency based on CPU utilization only. For example, the ondemand governor transitions to the next higher or lower p-state if the current CPU utilization crosses a threshold for a certain period of time. However, this approach can cause over- or under-provisioning as it is oblivious to the actual service performance. We argue that to save power without SLO violations, DVFS decisions should be based on both application-level performance and server power usage.

The goal of this work is to add another dimension to the power minimization problem that combines a commonly used approach for making applications elastic—adjusting services' capacity (CPU and memory) as well as CPU frequency scaling for the purpose of minimizing power consumption while meeting performance targets. The combination of different resource dimensions was shown to be useful for power efficiency in our previous work that combined CPU frequency scaling with horizontal and vertical elasticity [9].

Current autoscaling techniques offered by Infrastructure as a Service (IaaS) providers, e.g. Amazon EC2 [10] are based on horizontal elasticity—changing the number of Virtual Machines (VMs). Using a VM as scaling unit is coarse grained and can cause unnecessary over-provisioning [11]. In contrast, vertical elasticity involves dynamically changing the resource allocation of a running VM, typically in terms of CPU and RAM. It is a promising solution [12], [13], thanks to its fine-grained resource allocation and rapid enactment—individual fractions of a core may be allocated to a VM for as short time as a few seconds. Vertical elasticity could thus enable efficient resource provisioning and power usage.

Interactive applications commonly consist of three tiers: a front-end presentation tier, a middle-tier business logic (BL) server, and a back-end data storage (DS) tier. Each tier may require different combinations of resources (such as CPU, memory or both) at different times to fulfill user requests. For example, for the experiment conducted on the RUBiS [14] application BL tier, we observe increases in both CPU and memory usage with an increase in load (we give sufficient resources for the BL tier VM and the DS tier VM, refer to Section V-A for full specification of the hardware, software, and virtualization techniques used). The more memory and CPU given, the more threads the web server can allocate and use, resulting in a higher throughput. The aim of applying vertical elasticity in this scenario is to determine the right amount of CPU and memory to allocte to the VM dynamically in order to meet its throughput target.

978-1-5090-1654-9/16 $31.00 © 2016 IEEE
DOI 10.1109/ICAC.2016.32

In order to analyze the energy efficiency of servers, it is important to know the achieved performance and server power usage at a particular CPU utilization and frequency. Table I shows the impact of changing CPU frequency on CPU and power usage for the experiment conducted on a RUBBoS [15] BL tier VM. We run the experiment by fixing the number of cores (5 cores) and the load while we change the CPU frequency. The table shows the change in CPU and power usage for a similar throughput of 858 reqs/sec when running the server at different CPU frequency. For example, for a 10% increase in power usage (from 1.4 GHz to 2.1 GHz) there is a 50% reduction in the number of cores used. This could, for instance, enable more VMs to be packed to create a more consolidated and highly utilized server. On the other hand, during periods of lower load this could reduce power by running the server at lower frequency without SLO violations.

Table. I: CPU utilization and power usage for different CPU frequencies for a fixed workload.

CPU frequency (GHz)	CPU usage (%)	Power usage (Watt)
1.4	72	209
1.5	68	211
1.7	54	216
1.9	46	223
2.1	36	232

Based on these observations, we design a controller that in runtime selects a power-efficient configuration that combines fine-grained CPU and memory allocation for the VMs running collocated applications and frequency scaling of the physical cores. In particular, the controller allocates the right amount of resources for each application and combined with CPU frequency scaling it achieves this objective by selecting a configuration that would result in the minimal power usage. Our optimization strategy adapts to changes in system dynamics. In summary, the contributions of this work are:

- Design of online performance and power model estimators to dynamically determine the relationship between application-level performance (and server power) and the various configurations of the system. Our adaptive model captures variations in system dynamics due to differences in operating regimes, workload conditions, and application types (Section III).
- Design of a controller that uses the model estimators to automatically determine a configuration that minimizes power usage while meeting performance targets. We formulate two optimization algorithms and compare their impact on performance, CPU, and power usage. We also show that an operator can tune the tradeoff between power and performance (Section IV).
- An evaluation of the potential of the proposed approaches using three popular cloud benchmarks, RUBiS, RUBBoS, and Olio [16]. In comparison to the Linux CPU governors [8], our approaches achieves the lowest power consumption while meeting the performance targets (Section V).

II. ARCHITECTURE

Fig. 1 shows the architectural framework of the proposed controller. The system under control is a virtualized server hosting multiple applications. The controller computes the optimal configuration in two steps:

In the first step, a *performance model estimator* expresses the relationship between the performance and resource consumption using the application's previous resource usage pattern. More specifically, the estimator computes model parameters for an application, which then are used to forecast the application resource (CPU and memory) requirements. The *power model estimator* updates a model that captures the relationship between server power and its resource usage.

In the second step, using the respective model parameter values from the performance and power model estimators, the *controller* determines the CPU frequency, the number of cores, and the amount of memory that should be allocated to each application. A high level function of each component depicted in Fig. 1 is described as follows:

- The *sensor* periodically measures the applications' performance. We currently measure performance in terms of throughput, but the sensor can be extended to also handle other performance metrics. The sensor also gathers CPU and memory usage of each application along with power usage. The performance statistics and server power usage are then sent to the performance and power model estimators respectively, for model re-computation.
- The *performance model estimator* describes the relationship between an application's resource allocation and its performance. It automatically learns a model for this relationship as described in Section III.
- The *power model estimator* describes a relationship between the server's resource allocation and its power usage. This estimator also periodically updates a model for the dynamic relationship under the current operating conditions. The updated parameters from the performance and power model estimators are sent to the controller to determine the optimal configuration.
- The *workload monitor* gathers information about the required application performance. It monitors all incoming requests to the applications and calculates the required throughput (reqs/sec). This performance target is used as a reference signal to the controller.
- The *controller* determines the optimal reconfiguration, if any, for the next control interval to minimize power consumption and meet the performance target. This decision is based on the received applications' performance targets from the workload monitor and estimated parameters from the model estimators, The new configuration may result in changes in CPU frequency, number of cores, and/or memory allocation.
- The *actuator* changes the CPU frequency, number of cores and/or memory. The Xen [17] scaling and power management modules are used to actuate the new configuration.

III. DESIGN OF MODEL ESTIMATORS

A. Power Model Estimator

The power model estimator describes a relationship between resource allocation and power usage of a server. The processor and memory are two of the largest consumers of power in today's servers [18]. Servers also have non-

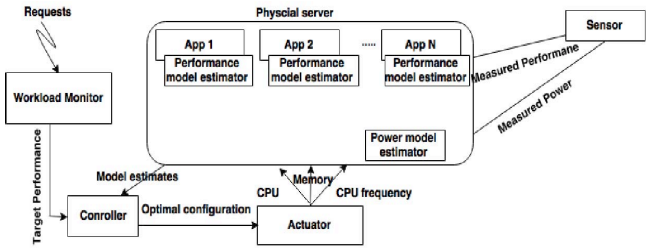


Figure. 1: Overall system architecture.

neglectable idle power usage. In this work, we model the server power usage using CPU and memory as described below.

1) CPU

An accurate power estimation can be obtained by considering the subunits within the processor that are active, specific instructions executed, on-chip cache usage, and frequency used. However, that requires a complete architectural model of the processor and has high processing overheads [18]. In this work, we use CPU utilization [6], [19] and CPU frequency [20] with associated CPU frequency voltage to model the CPU power usage. To obtain the CPU utilization, we adopt a lightweight method that tracks processor usage from the running VMs. Xen allows tracking the usage of the virtual CPUs from within a privileged domain, Domain0. By using this capability provided by Xen, we are able to extract physical CPU utilization from virtual processor usage. Based on this, the CPU power model for the server running N VMs is:

$$W^{\texttt{CPU}} = \beta^{\texttt{cpu}} \sum_{i=1}^{N} u_i^{\texttt{CPU}} f_i v_i^2 \tag{1}$$

where $u_i^{\texttt{CPU}}$ is the CPU utilization of VM$_i$, and f_i is the frequency with the associated voltage v_i of VM$_i$'s CPU. Note that the voltage depends on the frequency of the CPUs but to simplify the notation we do not explicitly write this dependency for the rest of the paper. The coefficient $\beta^{\texttt{cpu}}$ is a model-specific constant described in further detail below.

2) Memory

The key factor that affects memory energy usage is read and write throughput [18]. To capture memory throughput, we use the actual memory utilization of the VMs, which can easily be reported from within the guest domains. The memory power model for the server running N VMs is:

$$W^{\texttt{Mem}} = \beta^{\texttt{Mem}} \sum_{i=1}^{N} u_i^{\texttt{Mem}}, \tag{2}$$

where $u_i^{\texttt{Mem}}$ is the memory utilization of VM$_i$ and $\beta^{\texttt{Mem}}$ is a model-specific constant. The relationship between resource utilization and the actual power usage is normally nonlinear due to the complexity of computer systems. Since nonlinear control can lead to unacceptable runtime overhead, one feasible approach to capturing the system behavior is to linearize the system model [6], [21]. Here, we use the following linear model to approximate the quantitative relationship between the server resource (CPU and memory) and power usage:

$$\begin{aligned} W &= W^{\texttt{CPU}} + W^{\texttt{Mem}} + W^{\texttt{static}} \\ &= \beta^{\texttt{CPU}} \sum_{i=1}^{N} u_i^{\texttt{CPU}} f_i v_i^2 + \beta^{\texttt{Mem}} \sum_{i=1}^{N} u_i^{\texttt{Mem}} + W^{\texttt{static}}, \end{aligned} \tag{3}$$

where $W^{\texttt{static}}$ represents the system power usage not attributed to the varying CPU and memory usage. The parameters $\beta^{\texttt{CPU}}$ and $\beta^{\texttt{Mem}}$ are recomputed online using the Recursive Least Squares (RLS) method [6] for every control interval. The recursive nature of the RLS algorithm makes the time needed for this computation negligible.

B. Performance Model Estimator

The performance model estimator automatically learns and periodically updates a model for the dynamic relationship between the application's resource utilization and its performance. The performance of application i depends on the amount of cores and memory utilized as well as the CPU frequency of the VM. More precisely, we model the performance of application i as

$$P_i(u_i^{\texttt{CPU}}, u_i^{\texttt{Mem}}, f_i) = a_i^{\texttt{CPU}} u_i^{\texttt{CPU}} f_i + a_i^{\texttt{Mem}} u_i^{\texttt{Mem}}, \tag{4}$$

where $u_i^{\texttt{CPU}}$, f_i and $u_i^{\texttt{Mem}}$ are CPU utilization, CPU frequency, and memory utilization of VM$_i$ respectively. The coefficients $a_i^{\texttt{CPU}}$ and $a_i^{\texttt{Mem}}$ capture the relationship between the current performance and resource utilization. Here again, the parameters $a_i^{\texttt{CPU}}$ and $a_i^{\texttt{Mem}}$ are updated periodically using the RLS method.

IV. Controller Design

An issue with hardware support for power management in virtualized systems is that resources in a server are shared by multiple VMs [22]. For example, a physical core can be shared by many VMs at the same time. This means that the CPU frequency cannot be changed unless all VMs require the same frequency. We employ two approaches to overcome this limitation: i) *Fractional-CPU-per-VM*, a technique that considers uniform CPU frequency for all VMs but allow each VM to use fractions of cores and ii) *Frequency-per-VM*, a technique that allows each VM to have its own CPU frequency in an allocation that assigns a whole core(s).

Before providing detailed information about these approaches, we first introduce some common notation. The server comprises C cores and M GB of memory units. In total N virtual machines are running. Let $P_i^{\texttt{target}}(t)$ be the minimum performance target of VM$_i$ at time t, for $i = 1, 2, ..., N$. We let m_i and x_i denote the memory and the number cores allocated to VM$_i$, respectively, and we define $\boldsymbol{x} = (x_1, x_2, ..., x_N)^T$ and $\boldsymbol{m} = (m_1, m_2, ...m_N)^T$. Let $\boldsymbol{A_m} = \{\boldsymbol{m} \in \mathbb{R}^N \mid \sum_i m_i \leq M, m_i \geq 0 \forall i\}$ be the set of all possible memory allocations for the N virtual machines and let $\boldsymbol{A}_f$ be the set of clock frequencies available on the server.

A. Fractional-CPU-per-VM

The *Fractional-CPU-per-VM* algorithm considers uniform CPU frequency for the full physical machine for every control decision, i.e. all VMs run at the same CPU frequency. To run the VMs fast enough to meet their desired QoS, we use the fine-grained resource allocation capabilities provided by Xen scheduler. Here, the hypervisor scheduler attribute is used to determine the VM's maximum time slice. This allows cores to be allocated even at the fractional level.

The task of minimizing the power usage, at each time t, while attaining at least the pre-specified performance targets can now be formulated as the constrained optimization problem:

$$\min_{(\boldsymbol{x},\boldsymbol{m},f)\in\mathcal{A}} W(\boldsymbol{x},\boldsymbol{m},f)$$
subject to
$$P_i(x_i,m_i,f) \geq P_i^{\texttt{target}}(t) \;\; ,i=1,2,\ldots,N, \tag{5}$$

where the set of possible configurations is given by $\mathcal{A} = \{(\boldsymbol{x},\boldsymbol{m},f) \mid \boldsymbol{x} \in \boldsymbol{A_x}, \boldsymbol{m} \in \boldsymbol{A_m}, \text{and } f \in \boldsymbol{A}_f\}$. Here the set of feasible core allocations for N virtual machines, $\boldsymbol{A_x}$, is given by $\boldsymbol{A_x} = \{\boldsymbol{x} \in \mathbb{R}^N \mid \sum_i^N x_i \leq C, x_i \geq 0 \forall i\}$.

B. Frequency-per-VM

In this algorithm, a VM has exclusive access to a physical core for the full control interval, i.e. a VM is pinned to subset of cores. Hence, it is possible to change the CPU frequency of the physical cores of the VMs independently. The problem is to determine the number of cores, memory, and frequency for each VM in order to minimize the total power consumption while meeting time-varying performance constraints. To formulate this optimization problem, we let f_i denote the frequency of the cores allocated to VM$_i$ and we define $\boldsymbol{f} = (f_1, f_2, ..., f_N)^T$. Next, we define the following sets of admissible configurations. Let $\mathcal{X} = \{\boldsymbol{x} \in \mathbb{N}^N \mid \sum_i^N x_i \leq C\}$ be the set of all possible core allocations for the N virtual machines, $\mathcal{U} = \{(\boldsymbol{x},\boldsymbol{m},\boldsymbol{f}) | \boldsymbol{x} \in \mathcal{X}, \boldsymbol{m} \in \boldsymbol{A_m}$ and $\boldsymbol{f} \in \mathbb{R}^N, f_i \in \boldsymbol{A}_f \forall i\}$.

The task of minimizing the power usage, at each time t_n while attaining at least the pre-specified performance target can now be formulated as the constrained optimization problem:

$$\min_{(\boldsymbol{x},\boldsymbol{m},\boldsymbol{f})\in\mathcal{U}} W(\boldsymbol{x},\boldsymbol{m},\boldsymbol{f})$$
subject to
$$P_i(x_i,m_i,f_i) \geq P_i^{\texttt{target}}(t) \;\; i=1,2,\ldots,N. \tag{6}$$

C. Power-performance tradeoff

Although exactly corresponding to the desire of minimizing power consumption while attaining a predefined performance target, the formulations in Eqs. (5) and (6) are inflexible. In some cases, it might be acceptable to miss the performance target to save power[9]. We make our model more flexible by assigning a penalty for the deviation of the application's measured performance from its target. This way, performance lower than the target is feasible but becomes increasingly expensive. This strategy can be used to extend either techniques. Here, we illustrate for the *Fractional-CPU-per-VM* case. To make the overall minimization formulation dimensionally consistent, we translate the power consumption and performance penalty into monetary costs over the control interval $T = (\hat{t}, \hat{t}+\Delta t)$. In our model, we convert power usage to monetary costs as

$$C_T^{\texttt{Energy}} = y \int_T W(\boldsymbol{x},\boldsymbol{m},f)\,\mathrm{d}t, \tag{7}$$

where y is a constant representing the energy cost. In this work, a constant cost of power per time unit is considered but this can easily be changed to handle scenarios with varying costs.

There are multiple ways to relate performance penalty to monetary values and thus a variety of penalty functions can be chosen [23]. In this work, we use a *step-wise* penalty function as it provides an intuitive interpretation of SLAs and commonly used in the real-world contracts [24]. Here, the SLA penalty function is assumed to be a piecewise constant decreasing function of the performance conformance: $P_i(x_i,m_i,f)/P_i^{\texttt{target}}(t)$ of VM$_i$. This piecewise constant function is defined by k SLA penalty levels S_j, $j = 1,2,\ldots,k$, and $k+1$ conformance interval limits L_j, $j = 0,1,\ldots,k$, satisfying $0 = L_0 < L_1 < \ldots < L_{k-1} < L_k = \infty$. The cost of the SLO violation during the time interval T is given by:

$$C_T^{SLA} = \Delta t S_j, \tag{8}$$

where j is chosen such that:

$$L_{j-1} \leq P(x_i,m_i,f)/P^{\texttt{target}}(t) < L_j. \tag{9}$$

Incorporating this power-performance tradeoff, our minimization problem is formulated as:

$$\min_{(\boldsymbol{x},\boldsymbol{m},f)\in\mathcal{A}} \alpha C_T^{\texttt{Energy}} + \beta \sum_{j=1}^{K} C_T^{SLA}. \tag{10}$$

To allow human operators specify which of the two objectives (power and performance) is more important, we introduce two multipliers α and β, to represent priorities given to power savings and performance conformance, respectively.

V. Experimental Evaluation

A. Experimental setup

The experiments were performed on ProLiant DL165 G7 machines equipped with 32 AMD Opteron(TM) 6272 Processors and 56 GB of physical memory. The server has a CPU architecture with two sockets, two NUMA nodes per socket, and eight cores per NUMA node—32 cores in total. The server runs Ubuntu 14.04.2 with Linux kernel 3.13.0. The frequency of cores can be scaled from 1.4 GHz to 2.1 GHz.

To emulate a typical cloud environment and easily enable vertical elasticity, we use the Xen 4.4.1 hypervisor [17] with kernel version 3.13.0-49-generic. Hypervisors, such as Xen, have recently started supporting vertical scaling of CPU and memory. Unfortunately, existing memory *ballooning* techniques used to dynamically reallocate physical memory between VMs badly impact the performance of applications that manage their own memory, leading to thrashing and, in some cases, failure [25]. These applications include systems software like databases and language runtimes. Due to this reason, we do not apply vertical elasticity to the DS tier, we only do vertical scaling of the BL tier VM resources (number of cores, CPU capacity (%), and memory). Concerning memory scale-down, a VM may greatly be disturbed by reclaim of a large amount of memory as the inactive pages may not be ready to be reclaimed instantaneously. To let the VM shrink its memory gradually, we limit the maximum reduction to 20% of current memory size.

We use three different applications in our experiments: RUBiS, RUBBoS, and Olio. These applications are widely-used cloud benchmarks (see [11], [26]) and represent an eBay-like e-commerce, a Slashdot-like bulletin board, and a Web 2.0 social-event calendar applications, respectively. Each application was deployed on two separate VMs. VM1 runs the web tier with Apache 2.2.22 web server and PHP 5.3.10 application server. We used threadsafe Apache MPM prefork 2.2.22. VM2 runs the database tier with MySQL 5.5.43. We set different parameters regarding Apache processes, for example MaxClients and ServerLimit, to 2500, to handle the number of requests during any of our experiments. The DB

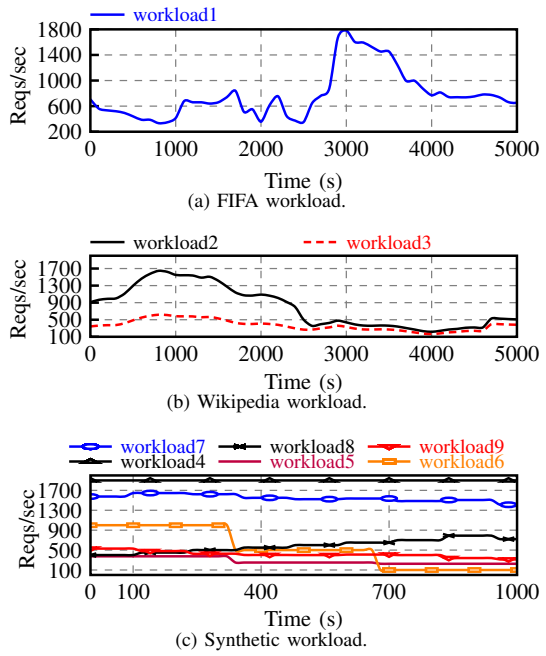


Figure. 2: FIFA, Wikipedia and synthetic workloads.

tier VMs run on a separate virtualized node. To avoid DB tier being a bottleneck, we assign sufficient resources—8 cores and 9 GB of memory—to satisfy the maximum load in our experiments. We also use separate nodes to run the clients for these applications. All nodes are in the same local network with a connection speed of 1Gb/s. RUBiS and RUBBoS BL tier VMs run on Ubuntu 12.04.3 LTS configured with 4 cores and 1.5 GB memory units initially. The Olio BL tier VM runs on Debian 7.3 with same initial core and memory units.

To emulate the users accessing the applications, we use the open source *httpmon*[1] tool with a closed-system client model. In this model a new client request is only issued after the previous request completes. We use the features available in this tool, such as think time and number of concurrent users, to stress-test the system. The think-time of each client was fixed at 0.5 second and the number of concurrent users (client threads) was varied dynamically during the course of the experiments. Furthermore, we instrument *lighttpd*[2] to monitor the number of incoming requests to each application. This information is used as a target throughput to evaluate the performance of the controller. As we use the incoming load as target throughput, the measured throughput can never be greater than the target except during transient states. Therefore, applications' reduced performance with respect to their target performance can only be due to shortage of resources, and not because of insufficient incoming load. To evaluate whether the controller can handle the dynamic variations in resource demands, we extract workloads based on traces from production environments, namely the FIFA [27] and Wikipedia [28] workload traces. These traces are chosen due to their complementary nature. While the Wikipedia workload shows a steady and predictable trend, the FIFA shows a bursty and an unpredictable trend. We also generate synthetic workloads to understand the behavior of the system that are not reflected under the real workloads. For example, the number of requests are incrementally decreased (workload 5 and 6) to study the effect of background load on interference. Fig. 2 shows the selected FIFA, Wikipedia, and synthetic workloads. To evaluate web-tier scalability, we use browse-only transaction patterns for all applications to apply high workload to the BL-tier and a low workload to DB tier. This prevents DB tier from being a bottleneck at any stage of the experiments.

The controller gathers monitoring information every 10 seconds using a light-weight monitoring framework. The monitoring framework (which implements the sensors for the controller) collects CPU utilization using Xen's DomainGetCPUStats interface. Memory utilization is collected using a background process that issues the $free$ command inside the VMs. More specifically, the memory usage of a VM is estimated by subtracting the free, cached and, buffered memory from the total memory. In order to leave room enough for the guest OS to properly operate, we also prevent the RUBiS, RUBBoS, and Olio VMs from being shrank below 1200, 1000, and 900 MB respectively. A rack-mounted HP AF525A Power Distribution Unit (PDU) meter is used for measuring power usages of the server. We use the Simple Network Management Protocol (SNMP) to extract power consumption of the server. We instrument the applications to gather performance metrics (measured throughput). This information is sent to the controller via UDP sockets. The actuator uses three mechanisms to make the change: i) Xen's credit-based CPU scheduler to assign cores and a cap (capacity in %) for each VM's CPU, ii) Xen's balloon driver for automated memory control, and iii) Xen's $xenpm$ tool to change the CPU frequency (P-States) of the cores. We also use $xenpm$ to switch from the *userspace* governor used by the controller to the governors used for comparison, the *ondemand*, and *performance* governors. Xen's default C-state C1 state is used for idle VCPUs to save some power without incurring greater wakeup latency [29].

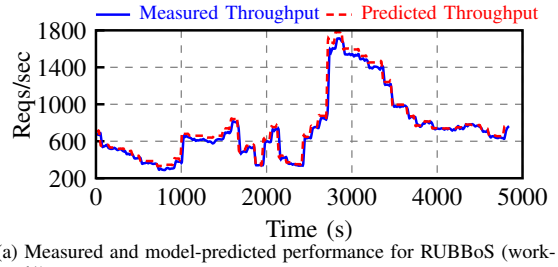
The controller is written in C and runs on Domain-0. A control interval of 10 seconds is used. We chose this interval since it is short enough to adapt the underlying infrastructure more quickly to the dynamic execution requirement of applications and long enough to observe the effects of the re-configuration [11], [21], and the overhead of actuation is negligible. To find appropriate parameters for the controller (see Eq. (7) - Eq. (10)), we set the power consumption cost based on a current electricity price of $0.2 per kWh over control interval T. The cost used for SLA penalty is inspired by existing SLAs for availability of various systems. The penality ($) doubles with a decrease in performance conformance (%); 100:99%–0.4$, 99:98%–0.8$,...,<90%–∞. The proposed optimization formulation is implemented using the Gurobi Optimizer 6.0 [30] solver for linear programming problems, which needs less than 10 ms to execute with the current setup.

B. Evaluation results

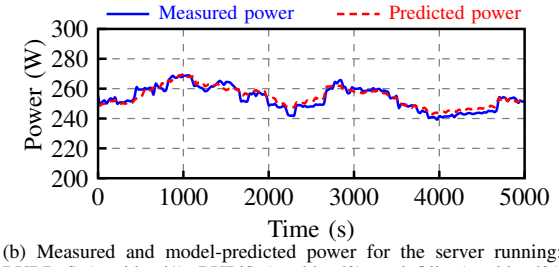
1) Model Estimator Accuracy

To assess the prediction accuracy of performance and power models, we use two validation measures—*coefficient of determination* (R^2) and *mean absolute percentage error*

[1]https://github.com/cloud-control/httpmon
[2]http://www.lighttpd.net/

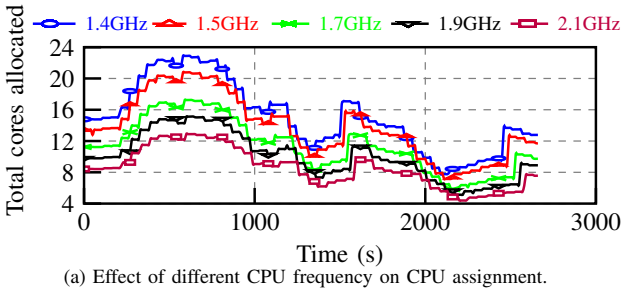


(a) Measured and model-predicted performance for RUBBoS (workload1).

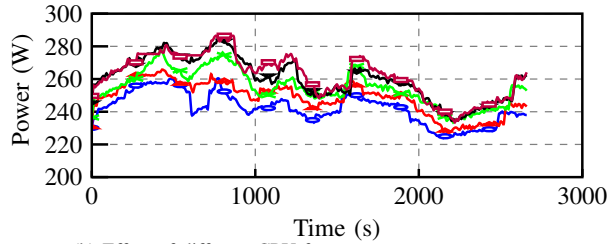


(b) Measured and model-predicted power for the server running: RUBBoS (workload1), RUBiS (workload2), and Olio (workload3.)

Figure. 3: Measured and model-predicted performance (RUBBoS) and power (server).



(a) Effect of different CPU frequency on CPU assignment.



(b) Effect of different CPU frequency on power usage.

Figure. 4: Effect of different CPU frequency on CPU assignment and power usage.

$(MAPE)$. These are defined as $R^2 = 1 - \frac{\sum_{i=1}^{n}(y_i - f_i)^2}{\sum_{i=1}^{n}(y_i - \bar{y})^2}$ and $MAPE = \frac{1}{n}\sum_{i=1}^{n}|\frac{(y_i - f_i)}{y_i}|$, where n represents the total number of samples, y_i, f_i and $\bar{y}$ represent the measured value, predicted value and the sample mean of y respectively. Table II shows the values of these two measures for the power model and the three applications' performance models. As an example, Fig. 3 shows the model-predicted and measured throughput for RUBBoS and power usage for the server. From these observations, we can see that a model when updated online can predict system dynamics with $MAPE$ below 10% and R^2 above 80%, which is considered sufficiently accurate [21].

Table. II: Prediction accuracy of performance and power models.

Model	$MAPE$	R^2
Power	1.8%	81%
RUBiS	4%	99%
RUBBoS	7.6%	95%
Olio	2.8%	98.2%

2) Effect of CPU Frequency on CPU and Power Usage

As described in Section I, the utilization of cores varies depending on the CPU frequency. To show how this observation is reflected in an efficient allocation of cores to VMs by our controller and how much power is used, we conduct an experiment using the *fractional-CPU-per-VM* algorithm. We use varying loads for VMs running RUBBoS, RUBiS, and Olio applications (workloads 1, 2, and 3 respectively). The experiment is repeated for each frequency with the same performance target for each application. Fig. 4 shows the core allocations and the corresponding power usage required to meet comparable performance targets under different frequency configurations. The core allocation is the sum of allocations for the three applications. From Fig. 4a, we observe that the controller allocates a decreasing number of cores for all applications when CPU frequency is increased. For example, when the server is running at 2.1 GHz CPU frequency, there is a total of 43% reduction in the total number of cores used compared to when running the server at 1.4 GHz, with a 7% increase in power consumption (see Fig. 4b).

In general, if more cores are available than what is required for the lowest CPU frequency, the controller runs the server at that frequency to reduce power the most. By doing this, it improves energy efficiency at lower server utilization. However, due to factors like increased load or allocation of new VMs, the number of cores required for a given frequency might exceed the cores available in the server. In that case the controller runs the server at a higher frequency in order to meet the performance targets. In this manner, more VMs can be consolidated to create a highly utilized server. All in all, these findings suggest that infrastructure providers could run more energy efficient servers by combining techniques that dynamically adjust server CPU frequencies with methods that provide fine-grained resource allocations.

3) Comparison with state-of-the-art techniques in terms of the power savings and performance guarantees

We compared the effectiveness of our techniques, *Fractional-CPU-per-VM* and *Frequency-per-VM*, in terms of reduced power usage against the Linux *performance* and *ondemand* CPU governors [8]. The *performance* governor keeps the server at full speed to handle peak load. This governor is used as a baseline. The *ondemand* governor manages the CPU frequency depending on system utilization: if current utilization is higher than an upper threshold (80%), the policy increases the frequency to the maximum. Whenever low utilization (less than 20%) is observed, the policy jumps directly to the lowest frequency that can keep the system utilization below 80%.

A *work-conserving* resource allocation method [21] is used for both *performance* and *ondemand* governors. In this allocation mode, the applications run in the default Xen settings, where a cap of zero is specified for the shared CPU, indicating

that they can use any amount of CPU resource. The purpose of choosing this allocation method is to evaluate the benefit of our method for power savings while meeting the performance targets compared to a scheme that only meets the later. For all experiments, the total amount of cores and memory assigned is less than the total available in the server to minimize the impact of resource contention among multiple running VMs.

Fig. 5 shows the results of these techniques in terms of performance of the three applications and power usage of the server. As shown in Figs. 5a to 5c all techniques meet the performance targets for RUBiS, RUBBoS and, Olio applications respectively. The main difference lies in power usage as shown in Fig. 5d. The *performance* governor consumes the most power since it sets the CPU statically to the maximum frequency and no dynamic optimization is performed. Even though the *ondemand* governor sets the CPU frequency depending on the current usage, it jumps to the maximum CPU frequency when the CPU utilization is above the upper threshold. As the incoming load to applications are taken as throughput targets, the measured throughput can never be greater than the target. This creates a best case scenario for the ondemand policy in terms of lower power consumption. In cases where the target is much lower than the incoming load, the policy enables an application to achieve a throughput much higher than its performance target. This will also results in a higher server power consumption. In addition, the governors are utilization-based, and are thus oblivious to the observed performance. Our techniques, on the other hand, takes into account the achieved throughput and the power consumed at a particular CPU as well as memory utilization and thus allocate resources to achieve performance targets while reducing power usage. *Fractional-CPU-per-VM* reduces power the most. The *Frequency-per-VM* consumes more power than *Fractional-CPU-per-VM* for the reason described in further detail below (Table III summarizes the power savings achieved by *ondemand*, *Fractional-CPU-per-VM* and *Frequency-per-VM* as compared to the baseline *performance* governor).

4) Comparison of Fractional-CPU-per-VM and Frequency-per-VM

Colocating workloads with coinciding resource demand peaks on modern hardware increases the risk of performance anomalies due to workload interference [31]. On multi-core, multiprocessor, and especially NUMA platforms, one approach to improve isolation is to pin certain applications to a subset of CPUs [3], [31]. To analyze the impact of pinning (and non-pinning) on performance and power consumption of colocated applications, we employ two configurations for the techniques under study. In the *Frequency-per-VM* technique each VM is constrained into a separate NUMA node. No explicit pinning of virtual CPUs to physical CPUs is enforced for the *Fractional-CPU-per-VM* technique. That is, the scheduler is free to schedule the virtual CPUs to arbitrary physical CPUs. We also vary the background load (with respect to one application this represents the total load of all colocated applications).

We performed experiments to demonstrate the differences between the two proposed techniques in terms of resources usage, power consumption, and their tolerance to interference.

Figs. 6a to 6c show the comparison in terms of resource usage for light background load (to achieve the same performance target). We use the same workloads and performance targets as in the experiments shown in Fig. 5. The memory allocation, shown in the bottom of the figures, is similar for both techniques. For CPU (CPU * Freq) allocation, shown in the top part, the unpinned configuration in the *Fractional-CPU-per-VM* technique makes use of fewer cores, which shows that the default scheduler uses CPUs efficiently. This observation is in line with findings in a recent study [31]—the unpinned configuration performs better than the pinned one in a system with light background load. Due to higher CPU usage, *Frequency-per-VM* consumes more power, as shown in Fig. 6d.

However, as the background load increases the *Frequency-per-VM* improves resource (such as cache) isolation, leading to more stable performance. This is shown in Fig. 8 with synthetic load. With a higher background load (workloads 5 and 6 for Olio and RUBBoS respectively) shown on top, RUBiS (using workload4) performs better under *Frequency-per-VM* than with *Fractional-CPU-per-VM*. As the background load decreases the performance improves for both techniques. Similar to Fig. 6d, the power consumption for *Frequency-per-VM* is higher (shown at bottom part of Fig. 8). A lesson from this is that systems hosting colocated workloads could benefit from dynamic selection of allocation techniques based on the system architecture, background load, and workload type.

Table. III: Impact of different techniques on power usage.

	Power savings (%)
Performance	-
Ondemand	6.7
Fractional-CPU-per-VM	16
Frequency-per-VM	6.3

5) Performance-Power tradeoff

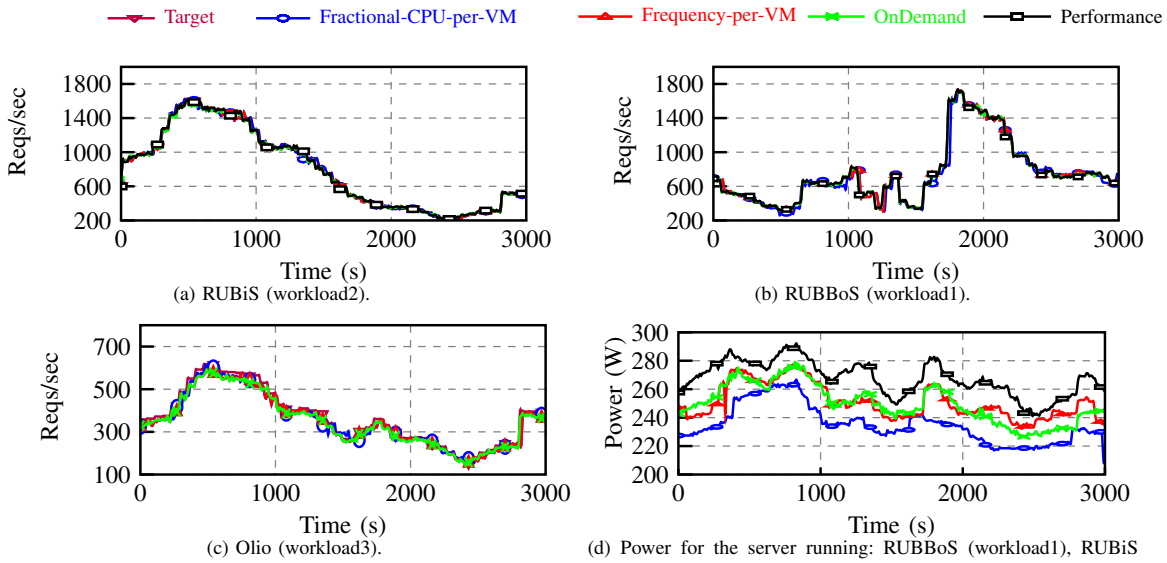
The extent to which the controller can reduce power depends on how much performance targets are violated. By varying the values of constants α and β in Eq. (10) we can steer the performance-power tradeoff as shown in Fig. 7. With a focus on power minimization (α= 0.9, β=0.1), the controller reacts quickly to minimize power usage. With a focus on performance (α=0.1, β=0.9), the controller responds more quickly to target performance deviations, resulting in higher power usage. Figs. 7a to 7d show up to 5% of power savings while meeting 92%, 90%, and 90% of the performance targets for the RUBiS, RUBBoS, and Olio applications respectively by setting α=0.9 and β=0.1. Table IV shows the performance deviation and power usage for the two weights. In general, the most suitable tradeoff between power saving and meeting performance target is a matter of operator preference and our controller simplifies handling of these conflicting goals.

Table. IV: Impact of performance-power tradeoff.

	Performance deviation	Average power
Performance priority ($\alpha = 0.1, \beta = 0.9$)		
RUBBoS	0.7%	246W
RUBiS	1.1%	
Olio	3%	
Power priority ($\alpha = 0.9, \beta = 0.1$)		
RUBBoS	10%	234W
RUBiS	7.7%	
Olio	9.9%	

VI. Related work

Significant research efforts have been expended on applying DVFS and VOVO to computing systems in order to



(a) RUBiS (workload2).
(b) RUBBoS (workload1).
(c) Olio (workload3).
(d) Power for the server running: RUBBoS (workload1), RUBiS (workload2), and Olio (workload3).

Figure. 5: Comparison of Fractional-CPU-per-VM, Frequency-per-VM, performance, and ondemand techniques in terms of performance and power consumption for the three concurrently running benchmarks with different workloads.

reduce power usage [32], [33], [34], [7]. Horvath et al. [32] present an approach based on DVFS and end-to-end delay control in multi-tier web servers. Sharma et al. [33] present a feedback loop that regulates frequency and voltage levels to meet deadlines. Chase et al. [34] consider how to reduce power usage in data centers by turning servers on and off based on resource demand. Niyato et al. [7] propose an optimal power management scheme to adjust the number of active servers for maximum energy savings. In contract to our work, these approaches are not designed for virtualized server environment and thus can not be applied in a multi-tenant infrastructure that host multiple services in the same server.

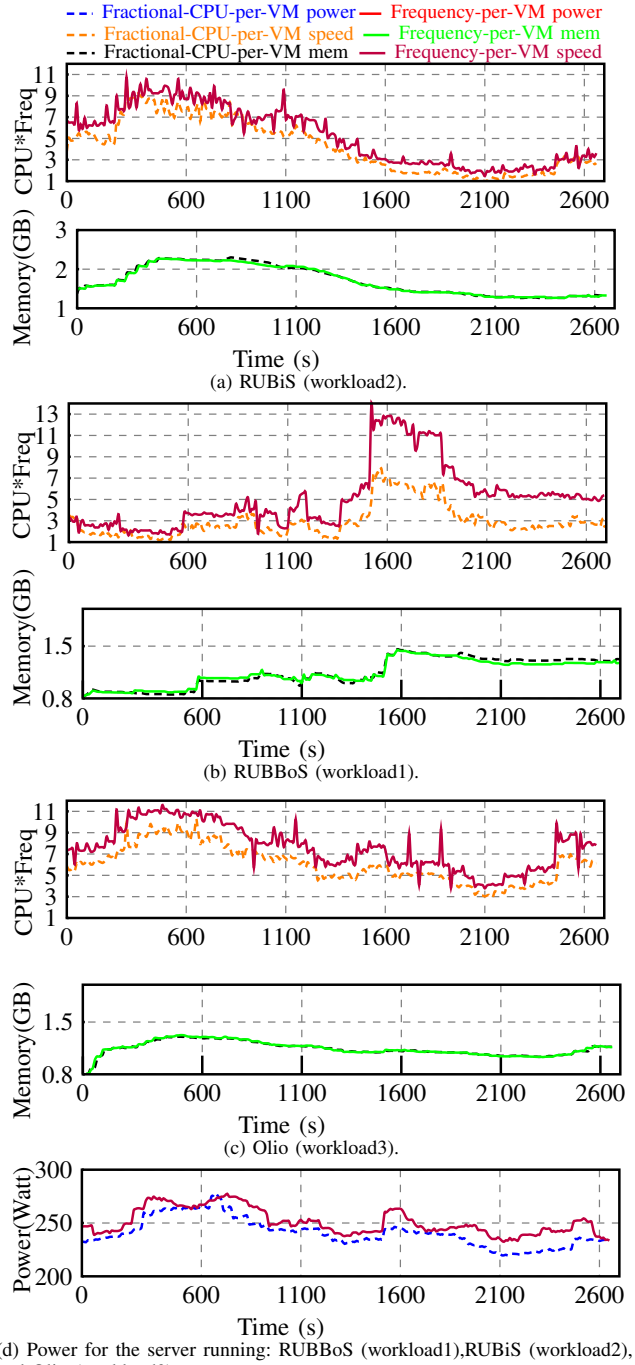
Dawoud et al. [13] compare vertical elasticity with respect to horizontal elasticity. They experimentally demonstrated that a fine-grained vertical elastic VM architecture consumes less resources and avoids scaling-up overhead while guaranteeing SLAs. Concerning CPU elasticity, Kalyvianaki et al. [35] design a controller using Kalman filters to control allocation based on CPU utilization. Padala et al. [21] applied a proportional controller to dynamically adjust CPU shares to VM-based multi-tier web applications. Work within memory elasticity includes Wang et al. [36] who propose a mechanism to dynamically set the amount of memory required to guarantee the performance of an application using sampling techniques. Molto et al. [37] present a mechanism for adjusting the VM memory size based on the memory usage pattern of the application using a simple elasticity rule. Diao et al. [38] design a MIMO controller to regulate server CPU and memory usage within specified QoS value. These works either focus on single configurable resource (CPU or memory) or consider both CPU and memory resource utilization as a decision making criteria which is oblivious to application performance [11]. Moreover, these works do not consider power usage in decision making. Stoess et al. [39] propose a power management framework in multi-layer virtualized system but their work ignores the system wide performance. In addition to considering power usage in decision making, our work focuses on vertical scaling of both CPU and memory resources and manages applications to meet performance guarantees.

In many works power usage is modeled as a function of CPU utilization [6], [19] while the CPU frequency is kept constant. Other works [40], [20] model power usage as a function of CPU frequency. Our work can contribute to a more realistic model for server power usage by combining the utilization of two of the most influential power consuming components—CPU and memory, as well as CPU frequency. In addition, the model parameters are dynamically estimated for more accurate management decisions. This applies to the performance model too, where changes in the performance model are detected and updated online.

Podzimek et al. [31] analyze impact of workload type, partial background CPU load, and CPU pinning configuration on performance interference and energy efficiency between pairs of colocated computationally-intensive workloads. They experimentally show that different CPU pinning configurations based on CPU load and workload type yields different resource isolation and utilization. While the main focus of their work is to investigate the impact of different configurations on performance and power usage, our work also make an optimized power and performance decision.

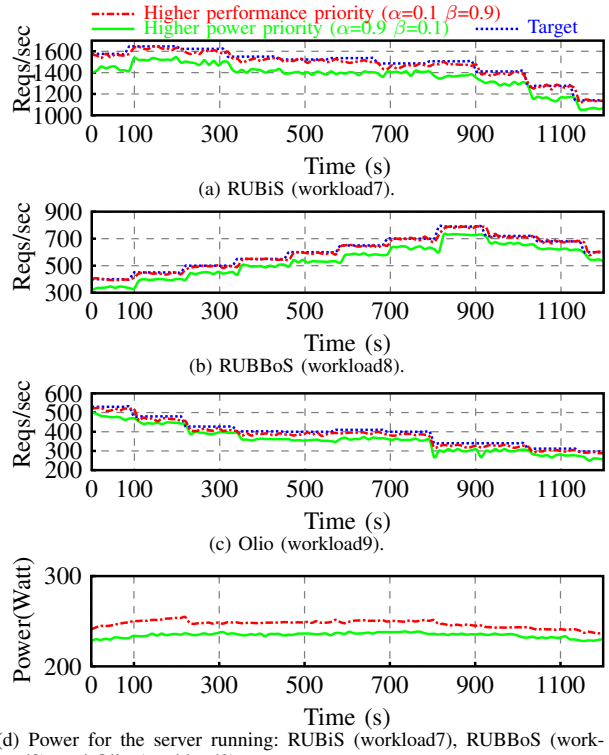
VII. Conclusion and Future Work

In this paper, we present an approach for power optimization at server level by dynamic configuration of the host and its VMs. We develop a power model and a performance model based on the number of CPUs, memory allocation, and CPU frequency. We adopt an approach that updates the model estimators online based on real-time measurements of performance and power usage, enabling system behavior to adapt to the needs of individual applications and different workload conditions. Leveraging the power and performance model estimators we devise two optimization strategies to determine the most power efficient configuration of the host and its VMs. On a system with light background load, the



(d) Power for the server running: RUBBoS (workload1),RUBiS (workload2), and Olio (workload3).

Figure. 6: Comparison of Fractional-CPU-per-VM and Frequency-per-VM in terms of resource and power usage for the three concurrently running benchmarks.



(d) Power for the server running: RUBiS (workload7), RUBBoS (workload8), and Olio (workload9).

Figure. 7: Impact of performance-power tradeoff on performance and server power usage for the three concurrently running benchmarks.

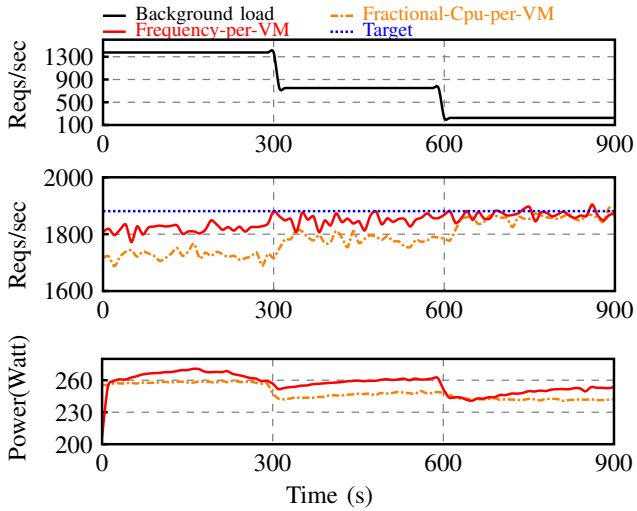


Figure. 8: Comparison of Fractional-CPU-per-VM and Frequency-per-VM in terms of interference.

Fractional-CPU-per-VM method enables more efficient utilization of resources and reduces power consumption. On a heavily loaded system, the *Frequency-per-VM* method improves CPU resource isolation, leading to more stable performance. However, the selection of technique, in general, should depend on the system architecture, and type and intensity of workload. We also show that human operators can tune the tradeoff between power and performance. An evaluation against the Linux *ondemand* and *performance* CPU governors shows the effectiveness of the proposed solution in power savings. Our solution achieves power savings from 12% to 20% compared to the baseline performance governor, while simultaneously meeting applications' performance goals.

For future work, we plan to integrate our solution in virtualized clusters to achieve both server level and the cluster level power optimization. We also plan to model and manage performance interference between consolidated VMs for efficient resource allocation control. We also would like to extend the work to include response time performance model to manage a richer set of services.

REFERENCES

[1] T. Mastelic, A. Oleksiak, H. Claussen, I. Brandic, J.-M. Pierson, and A. V. Vasilakos, "Cloud computing: Survey on energy efficiency," *ACM Comput. Surv.*, vol. 47, no. 2, Dec. 2014.

[2] V. Petrucci, O. Loques, and D. Mossé, "A dynamic optimization model for power and performance management of virtualized clusters," in *International Conference on Energy-Efficient Computing and Networking*, 2010, pp. 225–233.

[3] B. Subramaniam and W. Feng, "Towards energy-proportional computing for enterprise-class server workloads," in *SPEC*, 2013, pp. 15–26.

[4] C. Delimitrou and C. Kozyrakis, "Quasar: Resource-efficient and QoS-aware cluster management," in *ACM SIGPLAN Notices*. ACM, 2014, pp. 127–144.

[5] D. Lo, L. Cheng, R. Govindaraju, L. A. Barroso, and C. Kozyrakis, "Towards energy proportionality for large-scale latency-critical workloads," in *International symposium on Computer architecture*. IEEE Press, 2014, pp. 301–312.

[6] Y. Gao, H. Guan, Z. Qi, B. Wang, and L. Liu, "Quality of service aware power management for virtualized data centers." *Journal of Systems Architecture - Embedded Systems Design*, vol. 59, no. 4-5, pp. 245–259, 2013.

[7] D. Niyato, S. Chaisiri, and L. B. Sung, "Optimal power management for server farm to support green computing," in *CCGrid*, 2009, pp. 84–91.

[8] V. Pallipadi and A. Starikovskiy, "The ondemand governor: past, present and future," in *Linux Symposium*, vol. 2, 2006, pp. 223–238.

[9] S. K. Tesfatsion, E. Wadbro, and J. Tordsson, "A combined frequency scaling and application elasticity approach for energy-efficient cloud computing," *Sustainable Computing: Informatics and Systems*, vol. 4, no. 4, pp. 205–214, 2014.

[10] (Nov 2013) Amazon:ec2 [online]. http://aws.amazon.com/ec2/.

[11] E. B. Lakew, C. Klein, F. Hernandez-Rodriguez, and E. Elmroth, "Performance-Based Service Differentiation in Clouds," in *CCGrid*, 2015, pp. 505–514.

[12] L. Schubert, K. G. Jeffery, and B. Neidecker-Lutz, *The Future of Cloud Computing: Opportunities for European Cloud Computing Beyond 2010*, 2010.

[13] W. Dawoud, I. Takouna, and C. Meinel, "Elastic virtual machine for fine-grained cloud resource provisioning," in *Global Trends in Computing and Communication Systems*, 2012, pp. 11–25.

[14] (Feb 2014) Rubis [online]. http://rubis.ow2.org.

[15] (Feb 2014) Rubbos [online]. http://jmob.ow2.org/rubbos.html.

[16] (Feb 2014) Olio [online]. http://incubator.apache.org/projects/olio.html.

[17] P. Barham, B. Dragovic, K. Fraser, S. Hand, T. Harris, A. Ho, R. Neugebauer, I. Pratt, and A. Warfield, "Xen and the art of virtualization," *SIGOPS Oper. Syst. Rev.*, vol. 37, no. 5, pp. 164–177, Oct. 2003.

[18] A. Kansal, F. Zhao, J. Liu, N. Kothari, and A. A. Bhattacharya, "Virtual machine power metering and provisioning," in *SoCC*, 2010, pp. 39–50.

[19] R. Raghavendra, P. Ranganathan, V. Talwar, Z. Wang, and X. Zhu, "No power struggles: Coordinated multi-level power management for the data center," in *SIGARCH Computer Architecture News*, vol. 36, no. 1. ACM, 2008, pp. 48–59.

[20] J. Li, K. Shuang, S. Su, Q. Huang, P. Xu, X. Cheng, and J. Wang, "Reducing operational costs through consolidation with resource prediction in the cloud," in *CCGrid*, 2012, pp. 793–798.

[21] P. Padala, K.-Y. Hou, K. G. Shin, X. Zhu, M. Uysal, Z. Wang, S. Singhal, and A. Merchant, "Automated control of multiple virtualized resources," in *ACM European conference on Computer systems*, 2009, pp. 13–26.

[22] R. Nathuji and K. Schwan, "VirtualPower: coordinated power management in virtualized enterprise systems," vol. 41, Oct. 2007, pp. 265–278.

[23] Y. C. Lee, C. Wang, A. Y. Zomaya, and B. B. Zhou, "Profit-driven service request scheduling in clouds," in *CCGrid*, 2010, pp. 15–24.

[24] H. J. Moon, Y. Chi, and H. Hacigumus, "SLA-aware profit optimization in cloud services via resource scheduling," in *SERVICES*, 2010, pp. 152–153.

[25] T.-I. Salomie, G. Alonso, T. Roscoe, and K. Elphinstone, "Application level ballooning for efficient server consolidation," in *EuroSys*, 2013, pp. 337–350.

[26] Z. Shen, S. Subbiah, X. Gu, and J. Wilkes, "CloudScale: Elastic Resource Scaling for Multi-tenant Cloud Systems," in *ACM Symposium on Cloud Computing*, 2011, p. 5.

[27] "A workload characterization study of the 1998 World Cup Web site," *Network*, vol. 14, no. 3, pp. 30–37, May 2000.

[28] (Oct 2014) Wikimedia page view statistics [online]. http://dumps.wikimedia.org/other/pagecounts-raw/.

[29] (Apr 2015) Xen power management [online]. http://wiki.xenproject.org/wiki/Xen_power_management.

[30] (June 2015) Gurobi: Optimizer [online]. http://www.gurobi.com/.

[31] A. Podzimek, L. Bulej, L. Y. Chen, W. Binder, and P. Tuma, "Analyzing the impact of cpu pinning and partial cpu loads on performance and energy efficiency," in *CCGrid*, 2015, pp. 1–10.

[32] T. Horvath, T. Abdelzaher, K. Skadron, and X. Liu, "Dynamic voltage scaling in multitier web servers with end-to-end delay control," *IEEE TC*, vol. 56, no. 4, pp. 444–458, 2007.

[33] V. Sharma, A. Thomas, T. Abdelzaher, K. Skadron, and Z. Lu, "Power-aware QoS management in web servers," in *RTSS 2003*, 2003, pp. 63–72.

[34] J. S. Chase, D. C. Anderson, P. N. Thakar, A. M. Vahdat, and R. P. Doyle, "Managing energy and server resources in hosting centers," in *ACM SIGOPS Operating Systems Review*, vol. 35, no. 5, 2001, pp. 103–116.

[35] E. Kalyvianaki, T. Charalambous, and S. Hand, "Self-adaptive and self-configured CPU resource provisioning for virtualized servers using kalman filters," in *ICAC*, 2009, pp. 117–126.

[36] Y. Wang, C. C. Tan, and N. Mi, "Using elasticity to improve inline data deduplication storage systems," in *Cloud Computing (CLOUD)*, 2014, pp. 785–792.

[37] G. Moltó, M. Caballer, E. Romero, and C. de Alfonso, "Elastic memory management of virtualized infrastructures for applications with dynamic memory requirements," *Procedia Computer Science*, vol. 18, pp. 159–168, 2013.

[38] Y. Diao, N. Gandhi, J. L. Hellerstein, S. Parekh, and D. M. Tilbury, "Using MIMO feedback control to enforce policies for interrelated metrics with application to the apache web server," in *NOMS*, 2002, pp. 219–234.

[39] J. Stoess, C. Lang, and F. Bellosa, "Energy management for hypervisor-based virtual machines." in *USENIX annual technical conference*, 2007, pp. 1–14.

[40] Y. Chen, A. Das, W. Qin, A. Sivasubramaniam, Q. Wang, and N. Gautam, "Managing server energy and operational costs in hosting centers," in *SIGMETRICS PER*, vol. 33, no. 1, 2005, pp. 303–314.