Power Optimization with Performance Assurance for Multi-tier Applications in Virtualized Data Centers

Yefu Wang and Xiaorui Wang
University of Tennessee, Knoxville, TN 37996
{*ywang38, xwang*}*@eecs.utk.edu*

***Abstract*—Modern data centers must provide performance assurance for complex system software such as multi-tier web applications. In addition, the power consumption of data centers needs to be minimized to reduce operating costs and avoid system overheating. Various power-efficient performance management strategies have been proposed based on dynamic voltage and frequency scaling (DVFS). Virtualization technologies have also made it possible to consolidate multiple virtual machines (VMs) onto a smaller number of active physical servers for even greater power savings, but at the cost of a higher overhead. This paper proposes a performance-controlled power optimization solution for virtualized data centers with multi-tier applications. While existing work relies on either DVFS or server consolidation in a separate manner, our solution utilizes both strategies for maximized power savings by integrating feedback control with optimization strategies. At the application level, a multi-input-multi-output controller is designed to achieve the desired performance for applications spanning multiple VMs, on a short time scale, by reallocating the CPU resources and DVFS. At the data center level, a power optimizer is proposed to incrementally consolidate VMs onto the most power-efficient servers on a longer time scale. Empirical results on a hardware testbed demonstrate that our solution can effectively achieve performance-assured power savings. Extensive simulation results, based on a trace file of 5,415 real servers, demonstrate the efficacy of our solution in large-scale data centers.**

I. Introduction

In recent years, power has become one of the most important concerns for enterprise data centers hosting thousands of high-density servers and providing outsourced business-critical IT services. A well-known approach to reducing power consumption is to transition the hardware components from high-power states to low-power states whenever performance allows. For example, a widely used power-efficient server design is to have run-time measurement and control of the desired application performance by adjusting the CPU power states using Dynamic Voltage and Frequency Scaling (DVFS). However, while this approach can effectively reduce the dynamic power of the system, it cannot minimize the system leakage power for maximized power savings.

Recently, many data centers have begun to adopt server virtualization strategies for resource sharing. Virtualization technologies such as VMware and Xen can consolidate applications previously running on multiple physical servers onto a smaller number of physical servers, effectively reducing the power consumption of a data center by shutting down unused servers. More importantly, live migration [3] allows the movement of a virtual machine (VM) from one physical host to another with a reasonably short downtime [7]. This function makes it possible to use server consolidation as an online management approach, *i.e.*, having run-time estimation of resource requirements of every VM and dynamically re-mapping VMs to physical servers using live migration. When a more power-efficient VM-server mapping is found, unused servers can be put into the sleep mode for reduced power consumption.

While power consumption must be minimized, an important goal of data center operators is to meet the service-level agreements (SLAs) required by customers, such as response time and throughput. SLAs are important to operators of data centers because they are the key performance metrics for customer service and are part of customer commitments. Therefore, it is important to guarantee the SLAs of the applications while minimizing the power consumption of the data center.

Guaranteeing the desired SLAs with minimized power consumption introduces several major challenges. First, complex system software, such as web applications, commonly has multi-tier installations. Thus, an application may span multiple VMs. This characteristic calls for advanced Multi-Input-Multi-Output (MIMO) control solutions to manipulate multiple VMs simultaneously. Second, web applications often face significant, unpredictable workload variations. To guarantee SLAs, a control solution must respond to a workload variation quickly by adjusting system resource allocation (*e.g.*, CPU time on each server). This cannot be achieved by migrating some VMs among the servers because a VM migration typically requires seconds, or even minutes, to finish. Finally, servers in a data center may be manufactured by different hardware vendors and have different power efficiencies, *i.e.*, some servers are more power-efficient than others. A power management solution must be able to utilize this kind of heterogeneity to further reduce the total power consumption.

It is important to integrate application-level performance control and data center-level server consolidation for maximized power savings due to two reasons. First, most existing work of VM placement assumes that the resource requirements of the VMs are known a priori or can be estimated through measuring the resource utilization. However, resource utilization commonly differs from resource requirements, especially when the server is overloaded. Thus, it is preferable to have an application-level performance controller that can dynamically allocate system resource in

response to application requirement variations. Second, a performance controller may fail to guarantee the application performance when the server is overloaded due to possible workload increase. Therefore, it is preferable to have a data center-level consolidation algorithm to dynamically re-map VMs to physical servers to resolve the overload problem.

In this paper, we propose an integrated management solution to minimize power consumption for virtualized data centers while providing application-level performance assurance. Each application-level performance controller adopts a MIMO control strategy to maintain the desired performance and reduce power consumption through DVFS and dynamic CPU resource reallocation. The data center-level power optimizer then consolidates the VMs onto the most power-efficient servers and places unused servers into the sleep mode for power savings on a much longer time scale, to amortize the migration overhead. Specifically, the contributions of this paper are four-fold:

- We design a performance controller for multi-tier applications running in one or more virtual machines based on MIMO control theory, and analyze the control performance.
- We design a power optimizer that consolidates VMs onto a smaller number of physical servers in the data center based on the resource requirements determined by the application-level performance controllers to achieve minimized power consumption.
- We introduce the system architecture of our integrated power management solution, and the implementation details of each component.
- We present both empirical results on a hardware testbed and simulation results to demonstrate that our solution can effectively reduce data center power consumption while achieving the desired response time for multi-tier applications hosted in the data center. In addition, we show that our solution outperforms a state-of-the-art baseline, pMapper [22].

The rest of the paper is organized as follows. Section II highlights the distinction of our work by discussing related work. Section III introduces the overall architecture of our power management solution. Section IV presents the modeling, design, and analysis of the response time controller. Section V discusses our power optimizer. Section VI describes the implementation details of our testbed and simulator. Section VII presents the results with Section VIII concluding the paper.

II. Related Work

Power consumption is one of the most important design constraints for high-density servers. The majority of the prior work has attempted to reduce power consumption by improving the energy-efficiency of individual server components [12]. Several research projects have successfully developed power management algorithms at the server level [13], the cluster level [1, 16, 20, 23], and the data center level [24]. More closely related to this paper, Heo et al. [6] developed a power optimization algorithm using DVFS and shutting down unused servers for large-scale applications spanning several non-virtualized servers with a load balancer. In contrast to their work, our solution provides even greater power savings by consolidating underutilized servers onto a smaller number of active servers.

Virtualization technology has provided a promising way to manage application performance by dynamically reallocating resources to VMs. Several management algorithms have been proposed to control application performance for virtualized servers [2, 11, 17, 18, 26, 27]. For example, Padala et al. [17] proposed to control throughputs for virtualized servers. In contrast, our algorithm controls response time, a user-perceived performance metric. In addition to performance assurance, we integrate DVFS with server consolidation for maximal power savings. Furthermore, most existing work assumes simple single-tier applications [2, 18, 26, 27] while our solution relies on novel MIMO control theory to deal with complex web applications that may span multiple VMs.

VM migration is an important tool for resource and power management in virtualized computing environments. Some prior work focuses on using migration to satisfy the resource requests of VMs [8, 10, 25, 28]. In contrast, our work takes the total power consumption of the cluster as the design goal and designs a power-efficient algorithm while guaranteeing the desired performance requirements. Several recent studies propose to solve the VM-server mapping problem for power savings [10, 19, 21, 22]. For example, Raghavendra et al. [19] used a greedy bin-packing algorithm as part of their coordinated control solution to minimize power consumption in virtualized data centers. This paper has two differences compared to their work. First, our solution provides response time guarantees to multi-tier applications while their work focuses on single-tier applications. Second, our solution dynamically re-allocates CPU resource among the VMs while their performance control relies only on DVFS. Thus, our solution can allocate CPU resource to VMs based on their needs, which saves additional power.

III. System Architecture

In this section, we provide a high-level description of our system architecture, which includes an application-level response time controller and a data center-level power optimizer.

As shown in Figure 1, our power management solution includes two levels. At the application level, for every application running in the data center, there is a performance controller that dynamically controls the performance of the application by adjusting the CPU resource (*i.e.*, fraction of CPU time) allocated to the virtual machines running the application. We choose to control the 90-percentile response time of each multi-tier web application as an example SLA

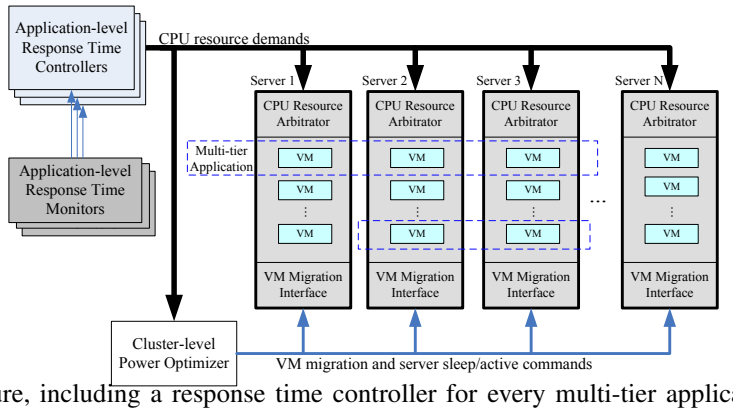


Figure 1: System architecture, including a response time controller for every multi-tier application and a data center-level power optimizer.

metric, but our management solution can be extended to control other SLAs such as average or maximum response times. We assume that the response time of a web application is independent from that of another web application. This is usually a reasonable assumption because they may belong to different customers. Hence, we choose to have a response time controller for each multi-tier application. A server-level CPU resource arbitrator then collects the CPU resource demands of all VMs hosted on the server, allocates the CPU resource to the VMs, and uses DVFS to save power, if the server has more CPU resources than the VMs require.

At the data center level, a power optimizer collects the CPU resource demands of all the VMs running in the data center, then uses a power optimization algorithm to find the most power-efficient VM-server mapping while satisfying the CPU resource requirements of all the VMs. The power optimizer then sends VM migration commands to the VM migration interfaces on every server, if necessary. It also puts selected servers into the sleep/active mode.

Server consolidation makes it possible to put unused servers into the sleep mode, which can typically save more power than DVFS. However, server consolidation may incur a higher overhead. Due to the change in workload, the power optimizer may require the migration of a VM from one server to another, or require a server to awaken. These operations are both time consuming, especially when the processors of the related servers are busy, or when the network bandwidth is limited. Thus, the optimizer should not be invoked too frequently. On the other hand, the response time controller at the application level uses CPU resource allocation and DVFS as its actuators, both of which have much smaller overheads than VM migration. As a result, the response time controller is invoked on a small time scale (several seconds) to deal with short-term variations in workload, while the power optimizer is invoked on a longer time scale (hours to days).

Between two consecutive invocations of the data center-level optimizer, it is possible that an unexpected increase of the workload can cause a severe overload on a server. To deal with this problem, the solution in this paper can be integrated with algorithms to move VMs from the overloaded servers to idle servers in an on-demand manner. An example of such algorithms can be found in our previous work [25].

IV. Response Time Controller

In this section, we present the problem formulation, modeling, and design of the response time controller.

A. Problem Formulation

Response time control can be formulated as a dynamic optimization problem. We first introduce some notation. A data center with N physical servers (S_1, S_2, ..., S_N) hosts L applications (App_1, App_2, ... , App_L). Some applications can be multi-tier applications running in multiple VMs. A VM running the j^{th} tier of App_i is named as VM_{ij}, $j = 1, 2, ..., r_i$. CPU allocation of VM_{ij} is defined as c_{ij}. Note that this number is expressed as the absolute number of CPU cycles allocated to VM_{ij} in terms of GHz. For example, If VM_{11} is allocated 20% of a 5GHz CPU, we say $c_{11} = 20\% \times 5GHz = 1GHz$. $\mathbf{c}_i = [c_{i1}, c_{i2}, ..., c_{ir_i}]^T$ is a column vector of the CPU allocations of all the VMs running application App_i. The 90-percentile response time of App_i is t_i seconds. The absolute CPU frequency of server S_i is f_i.

The control period of the response time controller is T seconds. Note that $x(k)$ denotes the value of x in the k^{th} control period, *e.g.*, $\mathbf{c}_i(k)$ means the value of $\mathbf{c}_i$ at time kT. Specially,$\mathbf{\Delta c}_i(k)$ is the difference between $\mathbf{c}_i(k+1)$ and $\mathbf{c}_i(k)$, *i.e.*, $\mathbf{\Delta c}_i(k) = \mathbf{c}_i(k+1) - \mathbf{c}_i(k)$.

At the end of every control period, the response time controller decides $\mathbf{\Delta c}_i(k)$ to achieve the desired response time for application App_i.

We assume that the constrained optimization problem is feasible, *i.e.*, there exists a set of CPU resource allocations within their acceptable ranges that can make the response time of the application achieve the desired value. If the problem is infeasible, *e.g.*, the application is highly I/O-intensive, no controller can guarantee the set points through CPU resource adaptation. In that case, the response time controller needs to be integrated with other resource allocation techniques, such as I/O scheduling, to achieve the desired response time, which is our future work.

B. Response Time Controller Design

In order to have an effective controller design, it is important to model the dynamics of the controlled system. The response time model of the application App_i is the relationship between t_i and $\mathbf{c}_i$. A well-established theoretical equation is usually unavailable for computer systems, therefore, we use a standard approach to this problem called *system identification* [5]. Rather than building a physical equation between the manipulated variables and the controlled variable, we infer their relationship by collecting data in experiments and then establish a statistical model based on the measured data. For example, by conducting system identification to an application in the prototype system introduced in Section VI, we get the system model as:

$$t_1(k) = \alpha_{11} t_1(k-1) + \boldsymbol{\beta}_{11}\mathbf{c}_1(k-1) + \boldsymbol{\beta}_{12}\mathbf{c}_1(k-2) + \gamma_1(k-1) \quad (1)$$

where α_{11}, $\boldsymbol{\beta}_{11}$ and $\boldsymbol{\beta}_{12}$ are constant parameters whose values can be determined in system identification.

We apply Model Predictive Control (MPC) theory [15] to design the controller, based on system model (1). MPC is an advanced control technique that deals with coupled MIMO control problems. This characteristic makes MPC well suited for response time control in multi-tier web applications.

A model predictive controller optimizes a cost function defined over a time interval in the future. The controller uses the system model to predict the control behavior over P control periods, called the prediction horizon. The control objective is to select an input trajectory that minimizes the cost function. An input trajectory includes the control inputs in the following M control periods, $\boldsymbol{\Delta}\mathbf{c}(k)$, $\boldsymbol{\Delta}\mathbf{c}(k+1|k)$, . . .$\boldsymbol{\Delta}\mathbf{c}(k+M-1|k)$, where M is called the control horizon. The notation $x(k+i|k)$ means that the value of variable x at time $(k+i)T$ depends on the conditions at time kT . Once the input trajectory is computed, only the first element $\boldsymbol{\Delta}\mathbf{c}(k)$ is applied as the control input to the system. At the end of the next control period, the prediction horizon slides one control period and the input is computed again based on feedback $t(k)$ from the response time monitor. Note that it is important to re-compute the control input because the original prediction may be incorrect due to uncertainties and inaccuracies in the system model used by the controller. MPC enables us to combine performance prediction, optimization, constraint satisfaction, and feedback control into a single algorithm. The controller includes a least squares solver, a cost function, a reference trajectory, and a system model. At the end of every control period, the controller computes the control input $\boldsymbol{\Delta}\mathbf{c}(k)$ that minimizes the following cost function:

$$J(k) = \sum_{i=1}^{P} \|t(k+i|k) - ref(k+i|k)\|^2_{\mathbf{Q}} + \sum_{i=0}^{M-1} \|\boldsymbol{\Delta}\mathbf{c}(k+i|k)\|^2_{\mathbf{R(i)}} \quad (2)$$

where P is the prediction horizon, and M is the control horizon. Q is the *tracking error weight*, and $\mathbf{R(i)}$ is the *control penalty weight vector*. The first term in the cost function represents the *tracking error*, *i.e.*, the difference between the response time $t(k+i|k)$ and a reference trajectory $ref(k+i|k)$. The reference trajectory defines an ideal trajectory along which the response time $t(k+i|k)$ should change from the current value $t(k)$ to the set point T_s (*i.e.*, desired response time). Our controller is designed to track the following exponential reference trajectory so the closed-loop system behaves like a linear system.

$$ref(k+i|k) = T_s - e^{-\frac{T}{T_{ref}}i}(T_s - t(k)) \quad (3)$$

where T_{ref} is the time constant that specifies the system response speed. A smaller T_{ref} causes the system to converge faster to the set point but may lead to a larger overshoot.

By minimizing the tracking error, the closed-loop system will converge to the response time set point T_s if the system is stable. The second term in the cost function (2) represents the *control penalty*, which causes the controller to decrease the change of the control input, *i.e.*, the CPU allocation. The control weight vector, $\mathbf{R(i)}$, can be tuned to represent a preference among the VMs. For example, a higher weight may be assigned to a VM if the process running it has a larger CPU demand so that the controller can give preference to increasing its CPU allocation.

In optimal control theory [14, 15], the stability of an MPC controller can be ensured by adding a *terminal constraint*. The constraint forces the state to take a particular value at the end of the prediction horizon. Therefore, we add the terminal constraint to our optimization problem requiring the response time of the application to converge to the set point at the end of the prediction horizon:

$$t(k+M|k) = T_s \quad (4)$$

The response time controller determines the CPU resource demand of every VM. A server-level CPU resource arbitrator then allocates the CPU resource to the VMs hosted on the server based on their demands. Specifically, the arbitrator on each server collects the CPU resource demand of every VM hosted on the server, in terms of CPU cycles per second (GHz), decides what CPU frequency the server should have in order to satisfy the aggregated demands, and then throttles the processor of the server to the desired CPU frequency using DVFS.

V. VM Consolidation for Power Optimization

As discussed in Section III, a data center-level power optimizer is used to find the most power-efficient VM-server mapping, and reconfigure the data center by VM migration. The optimizer then dynamically places selected servers into the sleep mode or wakes up selected servers for maximum power savings and guaranteed application performance.

The optimization problem falls in the category of *vector-packing* problems which are known to be NP-hard [10].

Therefore, we propose a heuristics-based optimization algorithm to find a polynomial time approximate solution.

Minimum slack problem for a single server: We begin with a sub-problem named the minimum slack problem. The problem can be presented as: given a server (not necessarily empty) and a list of unallocated VMs, select several VMs from the list, and allocate them to the server, such that the server has the least amount of unallocated CPU resource. This problem is a special case of the minimum bin slack (MBS) problem. Although it is an NP-hard problem, the MBS problem can be solved in pseudo-polynomial time [4]. The algorithm to solve the minimum slack problem is summarized in Algorithm 1. This algorithm is extended from the MBS algorithm in [4] by evaluating a more general constraint in each step, instead of checking if the total size of the items exceeds the size of the bin.

Power Aware Consolidation (PAC) for a list of servers: Another sub-problem, the power aware consolidation problem, can be presented as: given a list of servers (some servers are possibly not empty) and a list of VMs, consolidate the VMs to the servers, such that the total power consumption of the servers is minimized. Our proposed heuristics algorithm to solve this problem is described here. In the first step, the servers are sorted by power efficiency, *i.e.*, the ratio between the maximum CPU frequency and maximum power consumption of the server. Beginning from the most power-efficient server, we use Algorithm 1 to select several VMs from the remaining unallocated VMs, and then pack these VMs to this server such that the unused CPU resource in this server is minimized. We repeat this process with the next most power-efficient server until every VM in the list is allocated to a server.

Incremental Power Aware Consolidation (IPAC) algorithm: The PAC algorithm described above is invoked incrementally such that only a small number of VMs in a *migration list* are considered for consolidation each time. In each invocation period, some servers may be unable to host their VMs due to the possible workload increase. The algorithm first selects some VMs from these overloaded servers and adds them to the migration list to resolve the overload problem. Then, the VMs on the least power efficient server are added to the migration list. PAC algorithm is invoked to consolidate the VMs in the migration list to the servers. After the consolidation, if the number of active servers is reduced, PAC algorithm is invoked again to consolidate the VMs on the next least power efficient server until the number of active servers no longer decreases.

Cost-aware VM migration: The cost of the VM migration can be considerable. For example, if the network bandwidth is a bottleneck in a data center, a VM migration with high bandwidth consumption is the least preferred method. As a result, when the IPAC algorithm requests a migration, benefits and costs should be compared to decide if the migration should be allowed or rejected. The benefits of the migration include power savings. The cost of migration depends highly on the condition of the data center such as the network architecture, the bandwidth usage, the application itself and the memory usage of the VMs. Thus, the cost function can be highly different for different data centers. As a result, we provide an interface for data center administrators to define their own cost functions based on their various policies.

Algorithm 1 Minimum Slack

q: list of unallocated VMs
S: the server in consideration
ε: allowed slack
s^*: minimum slack
A^*: the collection of VMs best fits S
Minimum-Slack (q)
begin
1: **for all** VM VM_i in q **do**
2: Pack VM_i into S.
3: **if** S and the VMs allocated to it meets the constraint **then**
4: **if** Remaining CPU resource in S $< \varepsilon$ **then**
5: Exit;
6: **end if**
7: Minimum-Slack(q - VM_i);
8: **else**
9: Remove VM_i from S;
10: **end if**
11: **if** $Slack(S) < s^*$ **then**
12: $s^* = Slack(S)$
13: A^* =All VMs $\in S$
14: **end if**
15: **if** The algorithm does not finish in certain steps **then**
16: Increase ε by one step.
17: **end if**
18: **end for**
end

VI. System Implementation

In this section, we first introduce our testbed and the implementation details of each component. We then present the simulation environment used to test our PAC algorithm in various data center configurations.

A. Testbed

Our testbed includes a data center of four physical computers named $S1$ to $S4$. A fifth computer named *Storage* is used as the storage server for the Network File System (NFS) and is not part of the data center.

Xen 3.3 is used as the virtual machine monitor on all four servers in the data center. We use a PHP implementation of the RUBBoS benchmark [9], a bulletin board benchmark, as our server side workload. Each instance of RUBBoS is configured to be a two-tier application running in two VMs. The first tier has an Apache server installed and works as a

webserver running application scripts. The second tier has a MySQL database installed and acts as a database server.

The client-side workload generator is the Apache HTTP server benchmarking tool (ab). This tool allows users to manually define the concurrency level, which is the number of requests to perform in a short time, to emulate multiple clients. A concurrency level of 40 is used in our experiments to do system identification and most experiments, if not otherwise indicated. The workload generator runs on the Storage computer.

B. Simulator

To evaluate our power optimizer in large-scale data centers, we have developed a C++ simulator that uses a trace file from real-world data centers [24] to simulate the CPU utilization variations. The trace file includes the utilization data of 5415 servers from ten large companies covering the manufacturing, telecommunications, financial, and retail sectors. The trace file records the average CPU utilization of each server every 15 minutes from 00:00 on July 14th (Monday) to 23:45 on July 20th (Sunday) in 2008. We treat the utilization data of each server as the CPU demand of a VM. We generate 3000 simulated servers to host these VMs. Each server is randomly assigned one of 3 types of CPUs: $3GHz$ quad-core CPU, $2GHz$ dual-core CPU and $1.5GHz$ dual-core CPU.

VII. Experimentation

In this section, we present our empirical and simulation results. We first evaluate the response time controller and examine the power optimizer on the hardware testbed. We then present the simulation results of our power optimization algorithm in a data center with 5415 VMs.

We use pMapper, a heuristic-based algorithm proposed in a recent paper [22], as our baseline. PMapper is an incremental algorithm with two phases. In the first phase, it sorts the servers based on their power efficiency, then consolidates the VMs to the servers using a first-fit algorithm, beginning with the most power efficient server. Note that in this phase, the VMs are not actually migrated. In the second phase, pMapper computes the list of servers that require a higher utilization in the new allocation, and labels them as receivers. For each donor (servers with a target utilization lower than the current utilization), it selects the smallest-sized applications and adds them to a VM migration list. It then runs first-fit decreasing (FFD) to migrate the VMs in the migration list to the receivers.

Several differences exist between pMapper and our IPAC algorithm. First, pMapper is adapted from FFD while IPAC is adapted from Minimum Slack. Typically, Minimum Slack provides a better solution in terms of power consumption, especially when facing constraints such as memory constraint, bandwidth constraint, etc. Second, although Minimum Slack generally has a greater overhead compared with FFD, the IPAC algorithm considers only a very small number of VMs each time, while pMapper considers all the VMs in the first phase. Therefore, IPAC requires less computational overhead. Third, IPAC is combined with our response time controller to save more power using DVFS. Thus IPAC provides further power savings. Finally, the response time controller combined with IPAC can provide desired performance assurance in response to short-term workload variations.

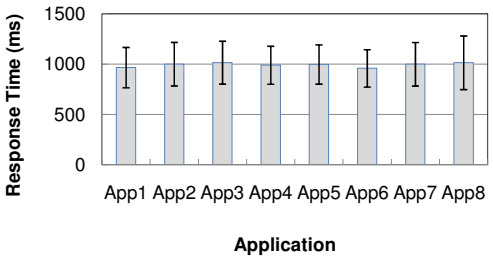
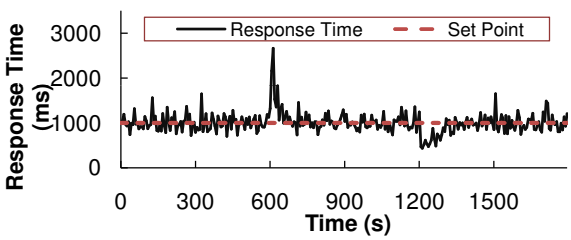


Figure 2: Response time of all 8 applications in the data center.

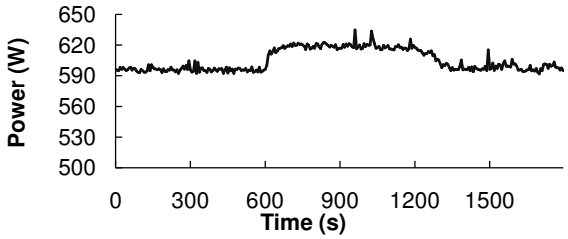
A. Response Time Control

In this experiment, we disable the power optimizer to evaluate the response time controllers of the 8 applications running in the data center. We first set the response time target for all applications to be $1000ms$. Figure 2 plots the the means and the standard deviations of the response times of the applications in the data center. This figure demonstrates that the response time controller works effectively to achieve the desired response time for all the applications. Figures 3(a) and (b) show a typical run of the response time controller for a randomly selected application, $App5$. At the beginning of the run, the controller achieves the desired response time set point, *i.e.*, 1000ms, after a short settling time. The workload of $App5$ increases significantly at a time of 600s. This is common in many web applications, *e.g.*, breaking news on a major newspaper website may incur a large number of accesses in a short time frame. To stress test the performance of our controller in such a scenario, we increase the concurrency level of $App5$ from 40 to 80 between time 600s and time 1200s to emulate the workload increase. The suddenly increased workload causes $App5$ to violate its response time limit at time 600s. The response time controller responds to the violation by allocating more CPU resource to the two VMs in both tiers. As a result, the response time of $App5$ converges to 1000ms again and the power consumption of the data center increases slightly due to the increased CPU resource usage, as shown in Figure 3(b).

To test the robustness of the response time controller when it is applied to a system that is different from the one used to do system identification, we conduct a set of experiments with wide ranges of concurrency levels. Figure 4 shows the average response times (with standard deviations) achieved by the controller when the concurrency level varies from 30 to 80. Figure 5 shows the average response times (with standard deviations) achieved by the controller when the



(a) Response time of App5 (response time controller)



(b) Power of the cluster (response time controller)

Figure 3: Typical runs of the response time controller and the baseline pMapper under a workload increase from time 600s to 1200s on App5.

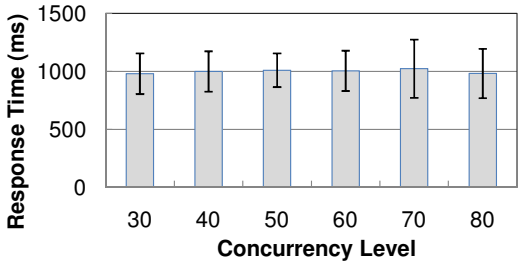


Figure 4: Response time of App5 under different workloads.

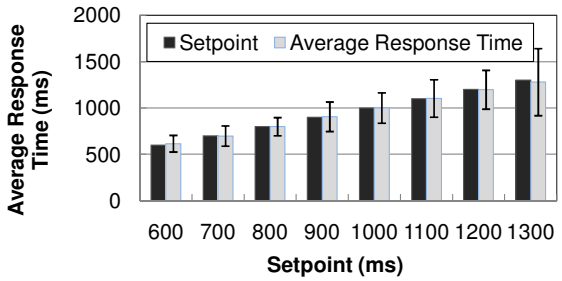


Figure 5: Response time of App5 under different set points.

response time set point increases from 600ms to 1300ms. The controller achieves the desired response time for all the currency levels and set points. The experiments demonstrate that the response time controller works effectively to achieve the desired response time when the actual system is under a workload different from the one used to design the controller.

B. Simulation Results in Large Data Centers

In this experiment, we test our power optimizer in large-scale data centers using the simulation environment introduced in Section VI-B.

We simulate 54 data centers with different number of VMs, ranging from 30 to 5,415. Every data center is assumed to have enough inactive servers which will be waken up and used if necessary. The parameters of the servers are introduced in Section VI-B. As an example of administrator-defined real world constraints, we add a restriction to the optimization algorithm such that the memory size of every server should be greater than the total memory allocations of the hosted VMs.

Figure 6 plots the average energy consumption per VM of IPAC and pMapper in 7 days under different number of VMs. In comparison to pMapper, IPAC shows lower energy consumption in all these simulations. On average, IPAC has a 40.7% more energy saving than pMapper. It is important to note that the energy savings of IPAC are due to two reasons. First, as discussed earlier, pMapper is adapted from FFD while IPAC is adapted from Minimum Slack. Typically, Minimum Slack provides a better solution in terms of power consumption . Second, IPAC is integrated with DVFS for power savings on a short time scale between two consecutive invocations of the optimization algorithm. Thus, IPAC saves more power when a lower CPU frequency is allowed due to short-term variations on the CPU requirements. Note that the testbed experiments in Section VII-A show a smaller power saving because only DVFS is tested and the CPU requirements of the workloads are relatively stable. With more VMs, the average energy consumption per VM becomes higher for both schemes. This is due to the fact that both algorithms try to use power-efficient servers first. With more VMs, more power-inefficient servers need to be used such that the per-VM power efficiency decreases.

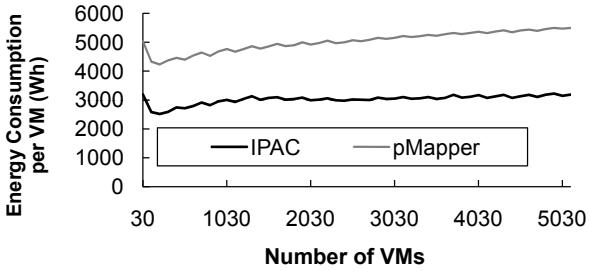


Figure 6: Energy consumption per VM in 7 days under different numbers of VMs.

VIII. Conclusions

In this paper, we have presented a performance-controlled power optimization solution for virtualized data center to achieve power efficiency and application-level performance assurance. While existing solutions rely on DVFS or server consolidation in a separate manner, our solution integrates feedback control with optimization to utilize both DVFS and server consolidation for maximized power savings with performance guarantees. At the application level, a MIMO controller is designed to achieve the desired 90-percentile response time for applications spanning multiple VMs, on a short time scale, by reallocating CPU resource and DVFS.

At the cluster level, a power optimizer is proposed to dynamically consolidate VMs onto the most power-efficient servers on a much longer time scale. Empirical results on a hardware testbed demonstrate that our solution can effectively achieve performance-assured power savings. Extensive simulation results, based on a trace file of 5,415 real servers, demonstrate the efficacy of our solution in large-scale data centers.

ACKNOWLEDGMENT

This work was supported, in part, by NSF Grants CNS-0720663, CNS-0845390, CNS-0915959, and ONR Grant N00014-09-1-0750.

REFERENCES

[1] L. Bertini, J. Leite, and D. Mosse. SISO PIDF controller in an energy-efficient multi-tier web server cluster for e-commerce. In *FeBID*, 2007.

[2] Y. Chen, S. Iyer, X. Liu, D. Milojicic, and A. Sahai. SLA decomposition: Translating service level objectives to system level thresholds. In *ICAC*, 2007.

[3] C. Clark, K. Fraser, S. Hand, J. G. Hansen, E. Jul, C. Limpach, I. Pratt, and A. Warfield. Live migration of virtual machines. In *NSDI*, 2005.

[4] K. Fleszar and K. S. Hindi. New heuristics for one-dimensional bin-packing. *Comput. Oper. Res.*, 29(7), 2002.

[5] G. F. Franklin, J. D. Powell, and M. Workman. *Digital Control of Dynamic Systems, 3rd edition*. Addition-Wesley, 1997.

[6] J. Heo, D. Henriksson, X. Liu, and T. Abdelzaher. Integrating adaptive components: An emerging challenge in performance-adaptive systems and a server farm case-study. In *RTSS*, 2007.

[7] M. R. Hines and K. Gopalan. Post-copy based live virtual machine migration using adaptive pre-paging and dynamic self-ballooning. In *VEE*, 2009.

[8] C. Hyser, B. McKee, R. Gardner, and B. J. Watson. Autonomic virtual machine placement in the data center, Tech Report, HP Labs. http://www.hpl.hp.com/techreports/2007/HPL-2007-189.pdf, 2008.

[9] Julie Marguerite and Emmanuel Cecchet. RUBBoS: Rice University Bulletin Board System. Rice University, http://www.cs.rice.edu/CS/Systems/DynaServer/RUBBoS/.

[10] G. Khanna, K. Beaty, G. Kar, and A. Kochut. Application performance management in virtualized server environments. In *NOMS*, 2006.

[11] D. Kusic, J. O. Kephart, J. E. Hanson, N. Kandasamy, and G. Jiang. Power and performance management of virtualized computing environments via lookahead control. In *ICAC*, 2008.

[12] C. Lefurgy, K. Rajamani, F. Rawson, W. Felter, M. Kistler, and T. W. Keller. Energy management for commercial servers. *IEEE Computer*, 36(12), 2003.

[13] C. Lefurgy, X. Wang, and M. Ware. Server-level power control. In *ICAC*, 2007.

[14] F. L. Lewis and V. L. Syrmos. *Optimal Control, Second Edition*. John Wiley & Sons, Inc., 1995.

[15] J. M. Maciejowski. *Predictive Control with Constraints*. Prentice Hall, 2002.

[16] R. Nathuji, P. England, P. Sharma, and A. Singh. Feedback driven qos-aware power budgeting for virtualized servers. In *FeBID*, 2009.

[17] 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 *EuroSys*, 2009.

[18] P. Padala, K. G. Shin, X. Zhu, M. Uysal, Z. Wang, S. Singhal, A. Merchant, and K. Salem. Adaptive control of virtualized resources in utility computing environments. In *EuroSys*, 2007.

[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 *ASPLOS*, 2008.

[20] K. Rajamani and C. Lefurgy. On evaluating request-distribution schemes for saving energy in server clusters. In *ISPASS*, 2003.

[21] A. Verma, G. Dasgupta, T. K. Nayak, P. De, and R. Kothari. Server workload analysis for power minimization using consolidation. In *USENIX*, 2009.

[22] A. Verma1, P. Ahuja, and A. Neogi. pMapper: Power and migration cost aware application placement in virtualized systems. In *Middleware*, 2008.

[23] X. Wang and M. Chen. Cluster-level feedback power control for performance optimization. In *HPCA*, 2008.

[24] X. Wang, M. Chen, and C. Lefurgy. SHIP: Scalable hierarchical power control for large-scale data centers. In *PACT*, 2009.

[25] X. Wang and Y. Wang. Co-con: Coordinated control of power and application performance for virtualized server clusters. In *IWQoS*, 2009.

[26] Y. Wang, X. Wang, M. Chen, and X. Zhu. Power-efficient response time guarantees for virtualized enterprise servers. In *RTSS*, 2008.

[27] J. Xu, M. Zhao, J. Fortes, R. Carpenter, and M. Yousif. On the use of fuzzy modeling in virtualized data center management. In *ICAC*, 2007.

[28] X. Zhu, D. Young, B. J. Watson, Z. Wang, J. Rolia, S. Singhal, B. McKee, C. Hyser, D. Gmach, R. Gardner, T. Christian, and L. Cherkasova. 1000 Islands: Integrated capacity and workload management for the next generation data center. In *ICAC*, 2008.