SHIP: A Scalable Hierarchical Power Control Architecture for Large-Scale Data Centers — Supplementary File

Xiaorui Wang, Ming Chen
Department of Electrical Engineering and Computer Science
University of Tennessee, Knoxville, TN 37996
{*xwang, mchen11*}*@eecs.utk.edu*

Charles Lefurgy, Tom W. Keller
Power-Aware Systems Group
IBM Research, Austin, TX 78758
{*lefurgy, tkeller*}*@us.ibm.com*

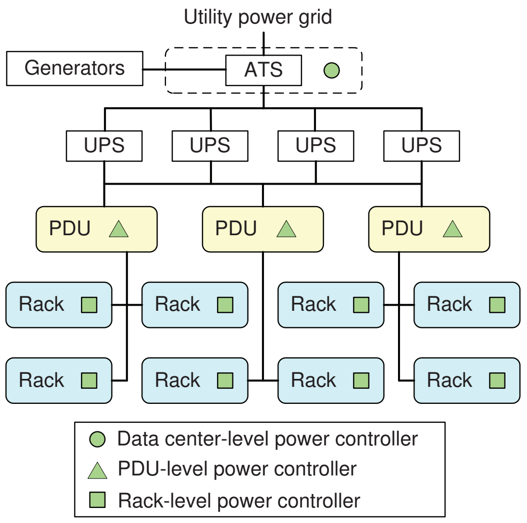

Utility power grid
Generators
ATS
UPS
UPS
UPS
UPS
PDU
PDU
PDU
Rack
Data center-level power controller
PDU-level power controller
Rack-level power controller

Fig. 1. Simplified power distribution hierarchy in a typical Tier-2 data center.

I. Introduction

The supplementary file of the paper is organized as follows. Section II introduces the power distribution hierarchy used in many data centers. Section III describes the controller design and analysis of the PDU-level power controller. Section IV discusses the rack-level power controller. Section VI provides additional implementation details of our control architecture and additional empirical results on a physical testbed. Section VII presents our simulation results in large-scale data centers. Section VIII discusses more related work. Section IX concludes the file.

II. Power Distribution Hierarchy

In this section, we mainly introduce the power distribution hierarchy used in many data centers. The power distribution hierarchy serves as a basis of our hierarchical power control architecture, which was presented in the paper.

Today's data centers commonly have a three-level power distribution hierarchy to support hosted computer servers [1], though the exact distribution architecture may vary from site to site. Figure 1 shows a simplified illustration of the three-level hierarchy in a typical 1 MW data center. Power from the utility grid is fed to an Automatic Transfer Switch (ATS). The ATS connects to both the utility power grid and on-site power generators. From there, power is supplied to Uninterruptible Power Supplies (UPS) via multiple independent routes for fault tolerance. Each UPS supplies a series of Power Distribution Units (PDUs), which are rated on the order of 75 - 200 kW each. The PDUs further transform the voltage to support a group of server racks.

A typical data center may house tens of PDUs. Each PDU can support approximately 20 to 60 racks while each rack can include about 10 to 80 computer servers. We assume that the power limit of the upper level (*e.g.,* the data center) is lower than the sum of the maximum power limits of all the lower-level units (*e.g.,* PDUs). This assumption is based on two key observations of data center operation. First, many data centers are rapidly increasing their number of hosted servers to support new business in the short term, while infrastructure upgrades at upper levels happen over much longer time scales due to cost considerations. Second, lower level units commonly have non-uniform workloads and so can rarely reach their power limits simultaneously. As a result, the cost of power infrastructure will soon be dominated by the upper-level units (PDUs and above).

III. PDU-level Power Controller Design and Analysis

In this section, we introduce the design and analysis of the PDU-level power controller. The problem formulation and system modeling are presented in the main paper. The data center-level controller is designed in the same way.

A. MPC Controller Design

We apply *Model Predictive Control* (MPC) theory [2] to design the controller. MPC is an advanced control technique that can deal with MIMO control problems with constraints on the plant and the actuators. This characteristic makes MPC well suited for power control in data centers.

A model predictive controller optimizes a *cost function* defined over a time interval in the future. The controller

The authors at the University of Tennessee are supported by NSF under a CSR grant CNS-0720663 and an NSF CAREER Award CNS-0845390, and by Microsoft Research under a power-aware computing award in 2008.

uses a 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 while satisfying the constraints. An input trajectory includes the control inputs in the following M control periods, $\mathbf{\Delta br(k)}$, $\mathbf{\Delta br(k+1|k)}$, … $\mathbf{\Delta br(k+M-1|k)}$, where M is called the *control horizon*. The notation $x(k+i|k)$ means that the value of variable x at time $(k+i)T_p$ depends on the conditions at time kT_p. Once the input trajectory is computed, only the first element $\mathbf{\Delta br(k)}$ 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 the feedback $pp(k)$ from the power 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 combines performance prediction, optimization, constraint satisfaction, and feedback control into a single algorithm.

The MPC 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 $\mathbf{\Delta br(k)}$ that minimizes the following cost function under constraints.

$$V(k) = \sum_{i=1}^{P} \|pp(k+i|k) - ref(k+i|k)\|^2_{Q(i)} + \sum_{i=0}^{M-1} \|\mathbf{\Delta br(k+i|k)} + \mathbf{br(k+i|k)} - \mathbf{P_{max}}\|^2_{\mathbf{R(i)}} \quad (1)$$

where P is the prediction horizon, and M is the control horizon. $Q(i)$ 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 total power $pp(k+i|k)$ and a reference trajectory $ref(k+i|k)$. The reference trajectory defines an ideal trajectory along which the total power $pp(k+i|k)$ should change from the current value $pp(k)$ to the set point P_s (*i.e.,* power budget of the PDU). Our controller is designed to track the following exponential reference trajectory so that the closed-loop system behaves like a linear system.

$$ref(k+i|k) = P_s - e^{-\frac{T_p}{T_{ref}}i}(P_s - pp(k)) \quad (2)$$

where T_{ref} is the time constant that specifies the speed of system response. A smaller T_{ref} causes the system to converge faster to the set point but may lead to larger overshoot. By minimizing the tracking error, the closed-loop system will converge to the power set point P_s if the system is stable.

The second term in the cost function (1) represents the *control penalty*. The control penalty term causes the controller to optimize system performance by minimizing the difference between the estimated maximum power consumptions, $\mathbf{P_{max}} = [P_{max,1} \dots P_{max,N}]^T$ and the new power budgets, $\mathbf{br(k+i+1|k)} = \mathbf{\Delta br(k+i|k)} + \mathbf{br(k+i|k)}$ along the control horizon. The control weight vector, $\mathbf{R(i)}$, can be tuned to represent preference among servers. For example, a higher weight may be assigned to a rack if it has heavier or more important workloads, so that the controller can give preference to increasing its power budget. As a result, the overall system performance can be optimized. In our experiments, we use the average CPU utilization of all the servers in each rack as an example weight to optimize system performance.

We have established a system model (3) for the PDU-level power consumption in Section III-B of the main paper. However, the model cannot be directly used by the controller because the system gains $\mathbf{G}$ are unknown at design time. In our controller design, we assume that $g_i = 1, (1 < i < N)$, *i.e.,* all the racks can achieve their desired power budget changes in the next control period. Hence, our controller solves the constrained optimization based on the following *estimated* system model:

$$pp(k+1) = pp(k) + [1 \dots 1]\,\mathbf{\Delta br(k)}. \quad (3)$$

In a real system that has different server configurations or is running a different workload, the *actual* value of g_i may become different than 1. As a result, the closed-loop system may behave differently. In Section III-B, we prove that a system controlled by the controller designed with the assumption $g_i = 1$ can remain stable as long as the actual system gain $0 < g_i < 14.8$. This range is established using stability analysis of the closed-loop system by considering the model variations. To handle systems with an actual g_i that is outside the established stability range, an online model estimator implemented in our previous work [3] can be adopted to dynamically correct the system model based on the real power measurements, such that the system stability can be guaranteed despite significant model variations.

This control problem is subject to the three constraints introduced in the main paper. The controller must minimize the cost function (1) under the three constraints. This constrained optimization problem can be transformed to a standard constrained least-squares problem. The transformation is similar to that in [4] and not shown due to space limitations. The controller can then use a standard least-squares solver to solve the optimization problem on-line. In our system, we implement the controller based on the `lsqlin` solver in Matlab. `lsqlin` uses an active set method similar to that described in [5]. The computational complexity of `lsqlin` is polynomial in the number of racks in the PDU and the control and prediction horizons. The overhead measurement of `lsqlin` can be found in [4].

B. Control Analysis for Model Variations

In this section, we analyze the system stability of the PDU-level power controller. A fundamental benefit of the control-theoretic approach is that it gives us confidence for system stability, even when the estimated system model (3) may change for different workloads. We say that a system is *stable* if the total power $pp(k)$ converges to the desired set point P_s, that is, $\lim_{k\to\infty} pp(k) = P_s$. Our MPC controller solves a finite horizon optimal tracking problem. Based on optimal control theory [6], the control decision is a linear function of the current power value, the power set point of the PDU, and the previous decisions for rack power budgets.

We now outline the general process of analyzing the stability of the PDU-level power consumption when the actual system model is different from the estimated model, *i.e.,* $g_i \neq 1$.

1) We compute the feedback and feedforward matrices for the controller by solving the control input $\mathbf{\Delta br(k)}$ based on the estimated system model (3) and the reference trajectory (2). The solution is in the following form:

$$\begin{aligned}\mathbf{\Delta br(k)} &= (\mathbf{K_{pp}} + \mathbf{K_v} + \mathbf{K_w})\mathbf{pp(k)}\\ &+ \mathbf{K_{\Delta br}}\mathbf{\Delta br(k-1)}\\ &+ \mathbf{K_{br}}\mathbf{br(k-1)} + \mathbf{H} + \mathbf{J}\end{aligned} \tag{4}$$

where $\mathbf{K_{pp}}$, $\mathbf{K_v}$, $\mathbf{K_w}$, $\mathbf{K_{\Delta br}}$, and $\mathbf{K_{br}}$ are parameter matrices. $\mathbf{H}$ and $\mathbf{J}$ are constant matrices that are independent of $\mathbf{pp(k)}$, $\mathbf{\Delta br(k-1)}$ and $\mathbf{br(k-1)}$. The designed MPC controller is a dynamic controller. Therefore, the stability analysis needs to consider the composite system consisting of the dynamics of the original system and the controller.

2) We then derive the closed-loop model of the composite system by substituting the control inputs derived in Step 1 into the actual system model (2) in the main paper. The closed-loop composite system is in the following from:

$$\begin{bmatrix}\mathbf{pp(k+1)}\\ \mathbf{\Delta br(k)}\\ \mathbf{br(k)}\end{bmatrix} = \begin{bmatrix}\mathbf{I} + \mathbf{G}(\mathbf{K_{pp}} + \mathbf{K_v} + \mathbf{K_w}) & \mathbf{G}\mathbf{K_{\Delta br}} & \mathbf{G}\mathbf{K_{br}}\\ \mathbf{K_{pp}} + \mathbf{K_v} + \mathbf{K_w} & \mathbf{K_{\Delta br}} & \mathbf{K_{br}}\\ \mathbf{0} & \mathbf{I} & \mathbf{I}\end{bmatrix} \begin{bmatrix}\mathbf{pp(k)}\\ \mathbf{\Delta br(k-1)}\\ \mathbf{br(k-1)}\end{bmatrix} + \begin{bmatrix}\mathbf{G} & \mathbf{G}\\ \mathbf{I} & \mathbf{I}\\ \mathbf{0} & \mathbf{0}\end{bmatrix}\begin{bmatrix}\mathbf{H}\\ \mathbf{J}\end{bmatrix}, \tag{5}$$

where $\mathbf{G} = diag\,[g_1, g_2, \ldots, g_N]$ and $\mathbf{I}$ is the identity matrix.

3) Derive the stability condition of the closed-loop system described by (5). According to control theory, if all poles are located inside the unit circle in the complex space, the system is stable. Solving this stability condition will give the range of g_i $(1 \le i \le N)$ where the system will guarantee stability.

Example. We now apply the stability analysis approach to the emulated PDU used in our experiments, which is composed of three server racks. We design the MPC controller by using the nominal system matrix $\mathbf{G} = diag\,[1, 1, 1]$. To analyze the system stability when the designed controller is used to control a different workload with a different system model, we derive the range of $\mathbf{G}$ in which all of the poles of the composite system are inside the unit circle.

We use three example ways to conduct this proof. First, we can assume that all the racks in the PDU have a uniform workload variation, *i.e.,* , $\mathbf{G} = g\mathbf{I}$. By following the steps stated above, we derive the range of g as $0 < g \le 14.8$. That means a system controlled by the MPC controller designed in our experiments can remain stable as long as its system parameters are smaller than 14.8 times of the values used to design the controller. Second, in the case that racks in a PDU commonly have different workload variations, our second way of analyzing the system stability is to assume that only one rack has workload variations at a certain time. We conduct the analysis for each rack and compute the range of g_i as $0 < g_i \le 42.6$. Finally, we investigate the case when one or more server racks have already reached their power constratins, i.e., $g_i = 0$. For example, if one rack is saturated, we have g_i as $\mathbf{G} = [0, 1, 1]$. The magnitude of the largest eigenvalue of the composite system is 0.4876, which is inside the unit circle. Hence, the system is stable.

Therefore, we have proven that a system controlled by our designed controller can remain stable even if it has three kinds of workload variations. The system stability with other workload variation patterns can be proven in a similar way. In addition, a Matlab program is developed by us to perform the above stability analysis procedure automatically. In our stability analysis, we assume the constrained optimization problem is feasible, *i.e.,* there exists a set of rack power budgets within the acceptable ranges that can make the PDU-level power consumption equal to its set point. If the problem is infeasible, no control algorithm can guarantee the power set point through power budget allocation. In that case, the system may need to integrate with other adaptation mechanisms (*e.g.,* workload redistribution among different racks). The integration of multiple adaptation mechanisms is part of our future work.

IV. Rack-Level Power Controller

In this section, we briefly introduce the rack-level power controller for the completeness of the paper.

The rack-level controller controls the power consumption of a server rack by manipulating the *CPU frequency* (and voltage) of the processors of each server in the rack. In this paper, we implement the rack-level power controller based on the algorithm presented in [7]. For the completeness of this paper, we briefly introduce its design as follows.

We first introduce some notation. T_r is the control period. $tp(k)$ is the total power consumption of all the servers in the rack. P_r is the power set point. $f_i(k)$ is the frequency level of the processors of Server i in the k^{th} control period. $\Delta f_i(k)$ is the frequency change, *i.e.,* $\Delta f_i(k) = f_i(k+1) - f_i(k)$. N_r is the total number of servers in the rack. The control goal is to guarantee that $tp(k)$ converges to P_r within a given settling time. The total power consumption, $tp(k+1)$, is modeled as:

$$tp(k+1) = tp(k) + \mathbf{A}\mathbf{\Delta f(k)} \tag{6}$$

where $\mathbf{\Delta f(k)} = [\Delta f_1(k) \ldots \Delta f_{N_r}(k)]^T$, $\mathbf{A} = [a_1 \ldots a_{N_r}]$, and $a_i, (1 \le i \le N_r)$ is a nominal control parameter determined based on system identification using a nominal workload, as introduced in [7]. The controlled system is proven to be stable even when a_i varies within $(0, 8.8a_i]$ due to different workloads or server configurations [7].

This controller is also designed based on MPC control theory. At the end of every control period, the controller computes the control input $\mathbf{\Delta f(k)}$ that minimizes the following cost function under constraints.

$$\begin{aligned}V_r(k) =& \textstyle\sum_{i=1}^{P_r} \|tp(k+i|k) - ref_r(k+i|k)\|^2_{Q_r(i)}\\ &+ \textstyle\sum_{i=0}^{M_r-1} \|\mathbf{\Delta f(k+i|k)} + \mathbf{f(k+i|k)} - \mathbf{F_{max}}\|^2_{\mathbf{R_r(i)}}\end{aligned} \tag{7}$$

where P_r is the prediction horizon and M_r is the control horizon. By minimizing the first term, the closed-loop system will converge to the power set point P_r if the system is stable. The second term causes the controller to optimize system performance by minimizing the difference between the highest frequency levels, $\mathbf{F_{max}}$, and the new frequency levels, $\mathbf{f(k+i+1|k) = \Delta f(k+i|k) + f(k+i|k)}$, along the control horizon. This control problem is subject to three constraints. First, the CPU frequency of each server should be within an allowed range (*e.g.,* Intel Xeon processor only has eight states). Second, two or more selected servers that run the same application service may need to have the same relative frequency level to avoid having a performance bottleneck. Third, the total power consumption should not be higher than the desired power constraint. The three constraints are modeled as:

$$F_{min,j} \leq f_j(k+1) \leq F_{max,j} \quad (1 \leq j \leq N) \tag{8}$$

$$f_i(k+1) = f_j(k+1) \tag{9}$$

$$tp(k+1) \leq P_r \tag{10}$$

Note that our control solution does *not* assume that servers run independent workloads, because the rack-level controller can handle coordinated applications running on multiple servers by configuring server CPU frequencies based on their application deployment [7]. Constraints can also be used to allow servers that run more important applications (*e.g.,* real-time tasks) to have higher CPU frequencies, and to prevent thermal emergency by lowering the frequencies of hot servers.

V. Discussion

The key advantage of power capping is that it provides a safe way for a data center to support more servers within the limited cooling and power supply capacities. As a result, data centers can gain a maximized return from their facility investment. In this section, we compare our hierarchical control solution with centralized solutions and discuss implementation issues.

A. Comparison with Centralized Solutions

The key advantage of the SHIP hierarchical control solution is that it decomposes the global control problem into a set of control subproblems at the three levels of the power distribution hierarchy in a data center. As a result, the overhead of each individual controller is bounded by the maximum number of units directly controlled by a controller, which is normally smaller than 100 in a typical data center, as discussed in Section II. Figure 2 shows that the average execution time of the MPC controller increases dramatically as the number of directly controlled servers increases. For example, the MPC controller with 100 servers takes approximately 0.39s but the MPC controller with 1,000 servers takes about 452.1s. Given that the control period of a data center-level power controller usually should be shorter than several minutes, a centralized MPC controller can only control a limited number of servers at the data center level. In addition, a centralized controller normally has undesired long communication delays in large-scale systems, resulting in degraded control performance. Therefore, centralized control solutions are not suitable to control an entire large-scale data center. A detailed comparison

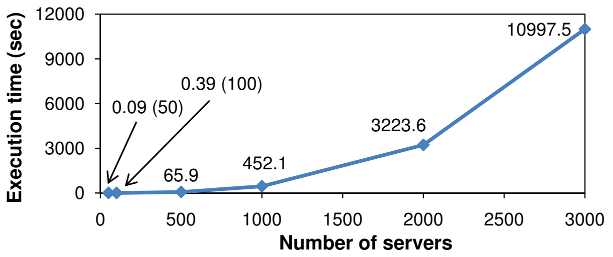


Fig. 2. Average execution time of the MPC controller for different numbers of servers.

between centralized and decentralized MPC control solutions can be found in [8].

Although a single centralized controller has poor scalability for an entire data center or PDU, centralized solutions can be used to control the power of each server rack *independently*. In our experiments presented in Section VI-C, we compare SHIP against a baseline called Per-Rack, which is composed of a set of state-of-the-art centralized MPC power controllers [7]. With Per-Rack, a separate controller is used to control the power of each rack independently from other racks. Since PDU- and data center-level power shifting is *not* the current practice in today's data centers, the power budget of each rack in Per-Rack is calculated in a static way by evenly dividing the total power budget of the PDU among all the racks in the PDU. Our results in Section VI-C show that that SHIP can allow different racks in a PDU to share power resource and thus outperforms Per-Rack because server racks commonly have non-uniform workloads.

B. Implementation Issues

Other challenges in implementing the proposed SHIP control hierarchy include communication delay and controller failure handling. In a data center, the rack-level controller can be implemented in service processor firmware in the rack enclosure with separate links to communicate with the main processor as in [9]. A PDU-level controller can also use communication links separated from data center networks to interact with rack-level controllers. Therefore, since controllers do not need to compete for bandwidth with data center workloads and the amount of data to be transmitted is small (just several numbers, such as power and utilization measurements and power budgets) in each control period, communication delay should not be a major problem. Furthermore, MPC control can tolerate a certain degree of communication delay as long as the delay is small compared to the control period [2]. If a communication link fails, the controller running on the firmware can detect the failure and then try to utilize alternative paths for communications (*e.g.,* via data center networks). To handle the failure of the hardware running the controllers, we could run a replica of each controller on a different service processor. This replica is a hot backup because the power and utilization measurements can be sent to both the primary and the backup controllers. The backup controller also conducts MPC control computation, just like the primary controller, but will not send out the control decisions for enforcement. As a result, the same states are maintained between the primary and the backup controllers.

When the hardware fails, sensors can detect that power control has failed (*e.g.,* broken communication or unexpected power violations). For example, some PDUs in today's data centers can measure branch circuit power and have the capability to send out an alert when the branch circuit is overloaded. Upon the receipt of the alert, the backup controller will replace the primary controller to enforce power capping. Please note that fault-tolerance is an active research area. Many more advanced fault-tolerance methodologies can be integrated with our control hierarchy to ensure the power safety of data centers. The detailed integration is out of scope of this paper.

VI. Additional Empirical Results

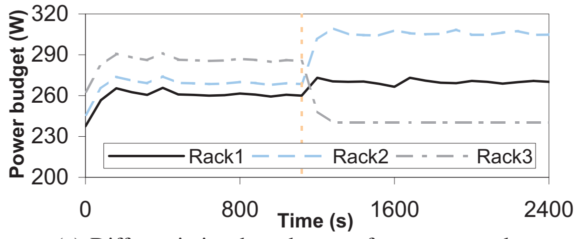
In this section, we present additional empirical results that are not presented in the main paper due to space limitations. We first examine the capability of SHIP to provide desired power differentiation. We then compare SHIP with a state-of-the-art centralized control solution.

A. Testbed Implementation

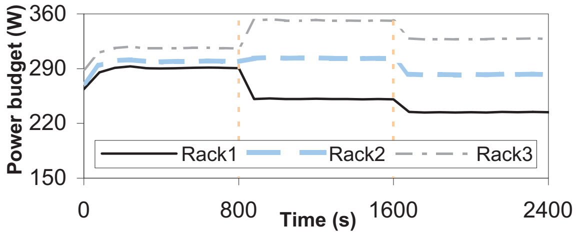
Our testbed includes 9 Linux servers to run workloads and a Linux machine to run the controllers. The 9 servers are divided into 3 groups with 3 servers in each group. Each group emulates a rack while the whole testbed emulates a PDU. Server 1 to Server 4 are equipped with 2.4GHz AMD Athlon 64 3800+ processors and run openSUSE 11.0 with kernel 2.6.25. Server 5 to Server 8 are equipped with 2.2GHz AMD Athlon 64 X2 4200+ processors and run openSUSE 10.3 with kernel 2.6.22. Server 9 is equipped with 2.3GHz AMD Athlon 64 X2 4400+ processors and runs openSUSE 10.3. All the servers have 1GB RAM and 512KB L2 cache. Rack 1 includes Server 1 to Server 3. Rack 2 includes Server 4 to Server 6. Rack 3 includes Server 7 to Server 9. The controller machine is equipped with 3.00GHz Intel Xeon Processor 5160 and 8GB RAM, and runs openSUSE 10.3. All the machines are connected via an internal Ethernet switch.

We now introduce the implementation details of the power controllers.

Power Controllers. Each rack-level power controller receives the power reading of the corresponding rack and the CPU utilizations of all the servers in the rack in the last control period. The controller then executes the control algorithm introduced in Section IV to compute new CPU frequency levels for the servers. The new levels are then sent to the CPU frequency modulators on the servers. The PDU-level controller receives the power measurement of the PDU and the average CPU utilizations of all the racks in the PDU in the last control period. The PDU controller then executes the control algorithm presented in Section III to compute new power budgets for the racks. The budgets are then sent to the rack-level controllers. The MPC controller parameters used in the experiments include the prediction horizon as 8 and the control horizon as 2. The time constant T_{ref}/T_p used in (2) is set as 2 to avoid overshoot while having a relatively short settling time.



(a) Differentiation based on performance needs.



(b) Proportional power differentiation

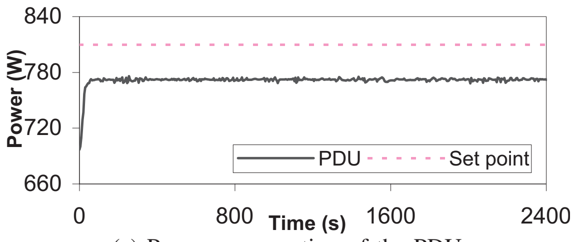
Fig. 3. Power budget differentiations.

B. Power Differentiation

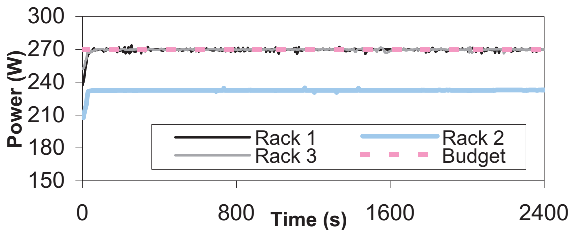
In many data centers, it is not uncommon for servers to have different power budgets, especially when the available power resource is not enough for all the servers to simultaneously run at their highest CPU frequency levels. For example, higher budgets can be given to servers running heavier workloads for improved overall system performance. In addition, power differentiation can be used to enforce desired priorities among servers. In this section, we evaluate the differentiation functions provided by the hierarchical control solution.

In the first experiment, we differentiate server racks by giving higher weights (*i.e.,* $\mathbf{R(i)}$) in the controller's cost function (1) to racks that have heavier workloads. Specifically, the weights are assigned proportionally to the racks' average CPU utilizations. Since running HPL on a server always leads to a 100% CPU utilization, we slightly modify the original HPL workload by inserting a sleep function at the end of each iteration in its computation loop, such that we can achieve different utilizations such as 80%, 50% for different servers. In the modified version of HPL, the problem size is configured to be $4,000 \times 4,000$ and the block size is set as 1. Note that the modified HPL benchmark is used *only* in this experiment. The power set point of the PDU is set to 810W. At the beginning of the run, we use the original HPL on all the servers so that all the racks have an average CPU utilization of 100%. As a result, all the racks are given the same weight. At time 1120s, we dynamically change the workload only on the servers in Racks 1 and 3 to run the modified HPL, so that the average CPU utilizations of Racks 1 and 3 become approximately 80% and 50%, respectively. Figure 3(a) shows that the controller responds to the workload variations by giving a higher power budget to a rack with a higher average CPU utilization. Rack 3 has the lowest budget because it has the lowest average utilization (*i.e.,* 50%). Note that application-level performance metrics such as response time and throughput can also be used to optimize power allocation in our solution.

In the second experiment, we differentiate server racks by enforcing a fixed ratio among the power budgets allocated to the racks, which is referred to as proportional power



(a) Power consumption of the PDU



(b) Power consumptions of the three racks

Fig. 4. A typical run of the Per-Rack power control solution.

differentiation. As power is becoming a first-class system resource, a commercial data center can use proportional power differentiation to give priorities to premium clients. As discussed in Section III-A of the main paper, constraint (2) can be used to achieve the desired fixed ratio. At the beginning of the experiment, we disable the constraint. As a result, the three server racks have a similar power budget allocation as before. At time 800s, we enable the power differentiation constraint to strictly enforce the power budget of Rack 3 to be 1.4 times that of Rack 1. As shown in Figure 3(b), in one control period, the power budget of Rack 3 rises to around 351.7W, which is exactly 1.4 times that of Rack 1 (251.2W). At time 1600s, the power set point of the PDU is dynamically reduced from 900W to 840W. Consequently, the budgets of all the racks are reduced. However, the desired budget ratio (*i.e.,* 1.4) has been maintained. Note that the fixed ratio can be strictly maintained only when all the budgets are within valid ranges.

The results of the two experiments demonstrate that SHIP can provide power differentiation for the consideration of system performance or client priorities. This property gives flexibility for data-center operators to effectively achieve desired differentiations in different scenarios.

C. Comparison with Per-Rack

In this experiment, we test SHIP and Per-Rack (introduced in Section V-A) in a special situation, where we keep the 3 servers in Rack 2 idling and run the benchmarks only on the 6 servers in Racks 1 and 3. Note that this situation is *not* uncommon in data centers because racks can be rented to different customers. As a result, some racks may have significantly more workloads than other racks at certain times. The power set point of the emulated testbed PDU is 810W. Under Per-Rack, each rack evenly gets 270W as its local power budget. Figure 4 shows that the idle rack (Rack 2) cannot use up its 270W power budget even when all its servers are running at their highest CPU frequency levels. On the other hand, Racks 1 and 3 cannot get enough power so that their servers can only run at degraded frequency levels. As a result, the power consumption of the PDU under Per-Rack is approximately 40W lower than the desired set point.

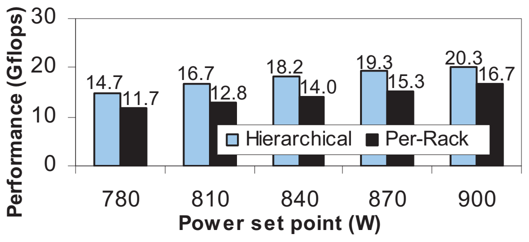


Fig. 5. Performance comparison between SHIP (Hierarchical) and Per-Rack using HPL.

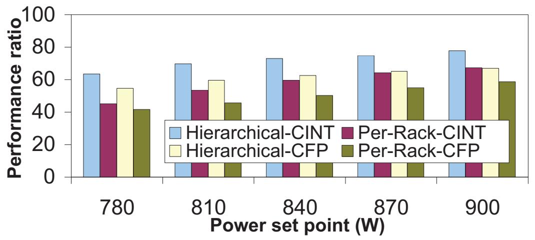


Fig. 6. Performance comparison between SHIP (Hierarchical) and Per-Rack using SPEC CPU2006.

Figure 5 shows the HPL performance data of SHIP and Per-Rack when the power set point changes from 780W to 900W. Each point (with the exact number on the top of each bar) is the sum of the HPL performance results of all the 6 servers in Racks 1 and 3. SHIP has better performance because its total power consumption can exactly reach the desired power set point. In contrast, the power budget of Per-Rack is wasted by the idle rack due to the even distribution of the total budget. The maximum performance improvement of the hierarchical solution is 30.5% at 810W. Figure 6 shows that SHIP outperforms Per-Rack also for the SPEC CPU2006 benchmarks. Note that we test this special situation just to highlight the performance improvement SHIP can achieve. When the workload difference of different server racks is not so significant, SHIP can still achieve performance improvement (probably less significant).

The experiments demonstrate that SHIP can allow different racks in a PDU to share power resource, which can lead to improved overall system performance because server racks commonly have non-uniform workloads.

VII. Simulations in Large-Scale Data Centers

We first introduce our simulation environment. We then test SHIP in large-scale data centers using a trace file from real-world data centers, which has the utilization data of $5,415$ servers.

A. Simulation Environment

To stress test the hierarchical control architecture in large-scale data centers, we have developed a C++ simulator that uses a trace file from real-world data centers to simulate the CPU utilization variations. The trace file includes the utilization data of $5,415$ servers from ten large companies covering manufacturing, telecommunications, financial, and retail sectors. The trace file records the average CPU utilization of each server in every 15 minutes from 00:00 on July 14th (Monday) to 23:45 on July 20th (Sunday) in 2008. We randomly generate several data center configurations based on the trace file. In each configuration, we group the servers into 6 to 8 PDUs with each PDU including 20 to 60 racks and each rack including 10 to 30 servers. Based on the specifications of the real servers used in our testbed, each server is randomly configured to have a minimum power consumption between 90W and 110W, a maximum power consumption between 150W and 170W, and a lowest relative frequency level between 0.3 and 0.5.

The simulator implements all the control components of the hierarchical control architecture, including the power and utilization monitors, controllers, and CPU frequency modulators. The controllers (at all the three levels) are implemented using functions provided by the Matlab libraries. At the beginning of each simulation, the simulator reads the data center configuration and initializes all the controllers accordingly. The power model of each rack is implemented based on (6) with the system parameter a_i calculated based on the randomly generated parameters including the maximum and minimum power consumptions and the minimum frequency level. The system parameter a_i of every server has a $\pm 5\%$ random variation in every control period to simulate workload variations in real systems.

In our simulations, all the servers start with the lowest DVFS level. In each control period of the rack-level loop, the utilization monitor of each server reads the utilization of the server from the trace file and sends the value to the rack-level power controller. The computed new CPU frequency level for each server is then used as the input of the power model (6) to calculate the power consumption of the rack in the next control period. The power consumption of each PDU is calculated by summing up the power consumptions of all the racks in the PDU. In each control period of the PDU-level loop, the power consumption of the PDU and the average utilization of each rack in the PDU are sent to the PDU-level power controller. The computed new power budgets are used as the set points for the rack-level controllers in the next control period. The data center-level control loop is implemented in the same way as the PDU-level loop.

B. Simulation Results

Figure 7 is a typical run of SHIP in a data center that is generated based on the method introduced in Section VII-A. This data center has 6 PDUs and 270 racks. The power set point of the data center is 750kW. As shown in Figure 7, the power of the data center precisely converges to the desired set point in two control periods of the data center-level control loop. Figure 8 plots the average power consumptions of three randomly generated data centers under a wide range of power set points from 600kW to 780kW. It is clear that SHIP can achieve the desired set point for the three large-scale data centers. The maximum standard deviation of all the data centers under all the power set points is only 0.72kW.

We then examine the capability of SHIP to differentiate PDUs based on the utilization data from the trace file. According to the controller design in Section III-A, the data center-level controller tries to minimize the difference between the estimated maximum power consumption and the power budget for each PDU. Therefore, a PDU with a higher average CPU utilization should have a smaller difference because of its higher weight in the controller's cost function. Figure 9(a) shows the average CPU utilizations of the 6 PDUs in the experiment, while Figure 9(b) shows the difference (*i.e.,* the estimated maximum power consumption minus the power budget) for each PDU. We can see that the difference order of the PDUs is consistent with the order of their average CPU utilizations. For example, PDU 2 has the highest average CPU utilization and thus the smallest difference. The results demonstrate that SHIP can effectively achieve the desired control objectives in large-scale data centers.

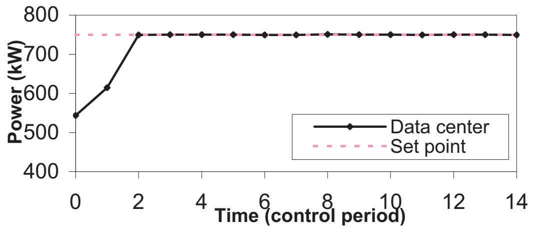


Fig. 7. A typical run of SHIP in a simulated large-scale data center.

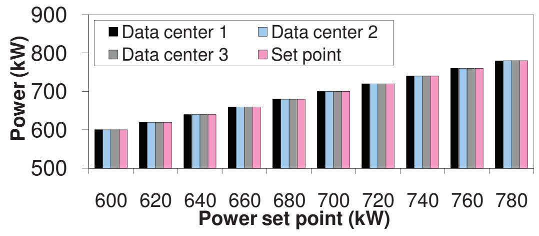
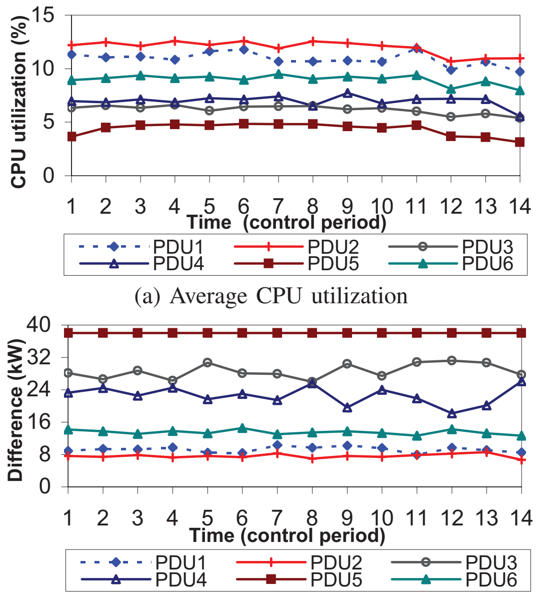


Fig. 8. Average power consumptions under different data centers and power set points.

VIII. More Related Work

In this section, we discuss more related work.

Control-theoretic approaches have been applied to a number of computing systems. A survey of feedback performance



(a) Average CPU utilization

(b) Difference between maximum power and budget

Fig. 9. Power budget differentiation based on average CPU utilizations.

control in various computing systems is presented in [10]. Feedback control scheduling algorithms have been developed for operating systems [11] and real-time systems [8]. Control techniques have also been applied to storage systems [12], networks [13], and Internet servers [14].

Some prior work has been proposed to use power as a tool for application-level performance requirements at the OS level. For example, Horvath et al. [15] use dynamic voltage scaling (DVS) to control end-to-end delay in multi-tier web servers. Sharma et al. [16] effectively apply control theory to control application-level quality of service requirements. Chen et al. [17] also present a feedback controller to manage the response time in a server cluster. Although they all use control theory to manage power consumption, power is only used as a knob to control application-level performance. As a result, they do not provide any absolute guarantee to the power consumption of a computing system. In this paper, we explicitly control the power consumption to adhere to a given constraint. Our solution is complementary to OS-level power management schemes and can be combined for increased adaptation capability and simultaneous control of power and system performance.

IX. Conclusions

This supplementary file presents additional details of the design and analysis of the PDU-level power controller, as well as additional empirical and simulation results of the paper: *SHIP: A Scalable Hierarchical Power Control Architecture for Large-Scale Data Centers*.

References

[1] X. Fan, W.-D. Weber, and L. A. Barroso, "Power provisioning for a warehouse-sized computer," in *ISCA*, 2007.

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

[3] Y. Wang, K. Ma, and X. Wang, "Temperature-constrained power control for chip multiprocessors with online model estimation," in *ISCA*, 2009.

[4] C. Lu, X. Wang, and X. Koutsoukos, "Feedback utilization control in distributed real-time systems with end-to-end tasks," *IEEE Transactions on Parallel and Distributed Systems*, vol. 16, no. 6, 2005.

[5] P. E. Gill, W. Murray, and M. H. Wright, *Practical Optimization*. Academic Press, London, UK, 1981.

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

[7] X. Wang, M. Chen, and X. Fu, "MIMO power control for high-density servers in an enclosure," *IEEE Transactions on Parallel and Distributed Systems*, vol. 21, 2010, to appear.

[8] X. Wang, D. Jia, C. Lu, and X. Koutsoukos, "DEUCON: Decentralized end-to-end utilization control for distributed real-time systems," *IEEE Transactions on Parallel and Distributed Systems*, vol. 18, no. 7, 2007.

[9] C. Lefurgy, X. Wang, and M. Ware, "Power capping: a prelude to power shifting," *Cluster Computing*, vol. 11, no. 2, 2008.

[10] T. F. Abdelzaher, J. Stankovic, C. Lu, R. Zhang, and Y. Lu, "Feedback performance control in software services," *IEEE Control Systems*, vol. 23, no. 3, pp. 74–90, Jun. 2003.

[11] D. C. Steere, A. Goel, J. Gruenberg, D. McNamee, C. Pu, and J. Walpole, "A feedback-driven proportion allocator for real-rate scheduling," in *OSDI*, 1999.

[12] M. Karlsson, C. T. Karamanolis, and X. Zhu, "Triage: Performance differentiation for storage systems using adaptive control." *ACM Transactions on Storage*, vol. 1, no. 4, pp. 457–480, 2005.

[13] S. Keshav, "A control-theoretic approach to flow control." in *Proceedings of ACM SIGCOMM*, 1991.

[14] J. L. Hellerstein, Y. Diao, S. Parekh, and D. M. Tilbury, *Feedback Control of Computing Systems*. John Wiley & Sons, 2004.

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

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

[17] Y. Chen, A. Das, W. Qin, A. Sivasubramaniam, Q. Wang, and N. Gautam, "Managing server energy and operational costs in hosting centers," in *SIGMETRICS*, 2005.