A Dynamic Energy Managment Scheme for Multi-tier Data Centers

***Abstract*—Multi-tier data centers have become a norm for hosting modern Internet applications because they provide a flexible, modular, scalable and high performance environment. However, these benefits come at a price of the economic dent incurred in powering and cooling these large hosting centers. Thus, energy efficiency has become a critical consideration in designing Internet data centers. In this paper, we propose a multi-facet approach consisting of dynamic provisioning, frequency scaling and dynamic power management (DPM) schemes to reduce the energy consumption of multi-tier data centers, while meeting the Service Level Agreements (SLAs). We formulate a mathematical model of the energy and performance/SLA optimization problem followed by a queuing theory based approach to develop two heuristics for solving the optimization problem.**

The first heuristic dynamically provisions the optimal number of servers required in each tier. The second heuristic proactively decides the CPU speed and the duration of sleep states of a server to achieve further energy savings. We evaluate our heuristics using a simulator that was validated with real measurements on a prototype three-tier data center consisting of 25 servers with two multi-tier application benchmarks. Our experimental results indicate that the proposed scheme, Hybrid can reduce the energy consumpion by 50%, relative to a statically provisioned data center that runs at the maximum CPU speed without utilizing the sleep state, while satisfying the SLAs with dynamically varying workloads. In addition, the proposed multi-facet approach is more energy efficient than the other existing schemes such as the dynamic provisioning with PowerNap [13].

I. INTRODUCTION

Multi-tier data centers are being increasingly used by Internet service providers for hosting modern applications with dynamic Web contents. Typical three-tier architecture with front-end Web servers, middle-level application servers and back-end database servers provides a modular, flexible and scalable environment for Web hosting [22]. However, the surging energy consumption of these data centers has become a serious concern from the economic and environmental standpoints. Server farms in U.S. are expected to consume 100 billion kWh at a cost of $ 7.4 billion per year by 2011 [24]. Service providers such as Google, Microsoft, Amazon, Akamai and Yahoo! spend millions of dollars annually to power on and cool their data centers. Therefore, instead of focusing on only high and scalable performance, the energy efficiency of data centers has become a first order goal to minimize the energy consumption for reducing the operating budget of data centers.

Most of the techniques for managing energy consumption in data centers fall into three broad categories. The first methodology involves dynamically turning on/off servers to save energy [2], [5]. The second approach uses Dynamic Voltage Frequency Scaling (DVFS), where a system dynamically adjusts the frequency/voltage to lower power usage [5]. The third technique uses Dynamic Power Management (DPM) [13], which utilizes sleep states of various components of a server system and decides when and for how long each component should be put to sleep to save power. Many proposed mechanisms combine some of the above techniques to enhance energy efficiency in data centers [2], [3], [15]. However, a systematic coordination of all the three techniques to boost the energy efficiency without sacrificing performance is not trivial. Since the first technique is employed at the global/system level while the DVFS and DMP are employed the local/component level as well as each technique is designed independently [16]. We are not aware of any previous work that has combined dynamic provisioning of number of servers, DVFS and DPM to investigate the performance energy trade-offs in data centers.

Dynamic provisioning of resources *i.e.*, allocation and deallocation of servers to applications through analysis of arriving workload patterns adapts well to the dynamic Internet traffic, compared to static provisioning [2], [22]. The authors in [3] propose a coordinated dynamic provisioning scheme with DVFS for a single-tier data canter, where a central controller decides the speed for the servers at different time frames and then the servers run at the same speed. The globally determined speed or sleep period might be reasonable, but it may not utilize local traffic patterns such as frequent, short idle periods reported in [13]. On the other hand, we may conservatively use a local energy management strategy such as immediately waking up a system from a sleep mode (DPM) as a request arrives [13]. However, it may not fully exploit the potential to advance energy efficiency without a global knowledge of the system. Therefore, we need to investigate how to coordinate dynamic provisioning with local energy management schemes, which is of interest in this paper.

In this work, we propose a three-prong approach, called Hybrid, for optimizing the energy consumption of multi-tier data centers, while satisfying the user specified SLAs. We develop a comprehensive mathematical model taking into consideration the performance (SLA) constraints and the energy/cost function. The model consists of two parts; global energy consumption of a data center and energy consumption of individual servers in each tier. Since the model becomes NP-complete, we propose two heuristics for solving the global and local optimizations based on queueing theory. The first heuristic, for global optimization, dynamically provisions the required number of servers across each tier. The number of servers is obtained by modeling the multi-tier data center as a queueing network and using the Mean Value Analysis (MVA) [18] for estimating the performance of each tier. The second heuristic proactively decides the CPU speed (DVFS) and the duration of a sleep state of a server (DPM) using the results from MVA for local energy optimization. The proposed scheme is extensively evaluated with a multi-tier data center

simulator which in turn is validated with real measurements in terms of energy consumption and response time by a prototype three-tier data center of 25 servers. We used two benchmarks, RUBiS [4] and TPC-W [20] for both validating our simulator and evaluating the heuristics.

The experimental results indicate that our proposed three-prong Hybrid solution consisting of dynamic provisioning, DVFS and DPM could save energy up to 50%, compared with the base case of static provisioning of a multi-tier data center without utilizing DVFS and DPM. We demonstrate the energy efficiency of the proposed approach compared to three other dynamic provisioning techniques; dynamic provisioning without local energy management, dynamic provisioning with DVFS, and dynamic provisioning with the DPM scheme in [13]. For the RUBiS workload, our approach provides additional energy saving by 46% compared to dynamic provisioning only and by 42% compared to the recently proposed PowerNap technique [13]. This significant amount of energy savings stems from the fact that the proposed scheme creates longer total sleep duration per host and provisions less number of servers to balance out the flash crowd. All the dynamic schemes exhibit similar energy behavior for the TPC-W workload since this workload generates many small requests, thereby making it difficult to utilize the local DVFS and DPM techniques effectively. We also show that the proposed scheme, Hybrid can provide substantial energy savings with little impact on performance since it is able to satisfy the required SLAs.

The rest of the paper is organized as follows: In Section II, we discuss the related work. Backgrounds behind the problem formulation are described in Section III. Section IV and V lay out the formulation of the problem and the methodology used for solving it. Our results and discussions are presented in Section VI followed by concluding remarks and future work in Section VII.

II. Related Work

Resource capacity planning and dynamic provisioning for QoS control have been explored in the past. Chase et al. [2] presented an approach using economic theory for resource management in hosting centers with emphasis on energy and SLA. They however considered only a single web tier with small number of web servers for their experiments and did not leverage the benefits of using DVFS and sleep states for saving energy. An analytical model of a multi-tier data center was proposed in [23]. Dynamic provisioning of a multi-tier data center has been explored in [22]. Both of the above works have focused on modeling and dynamic provisioning with respect to only performance (response time and throughput) of multi-tier data centers without considering their energy/power. One of the contributions of this paper is to consider dynamic provisioning for optimizing both performance and power/energy consumption in multi-tier data centers.

Power management in server systems adopts mostly three techniques - shutting down selected servers, modulating CPU operating frequency/voltage and dynamic power management, which puts a system to sleep states to save power. Initial studies [2], [15] have used the methodology of shutting down the servers that are either not in use, or have low load to conserve power. Subsequent studies [5], [19] have looked at optimizing power efficiency by monitoring system load and dynamically modulating the CPU frequency/voltage through DVFS. The authors in [5] have also used a combination of DVFS and turning on/off servers approaches for power management. Also, finding the required number of servers to minimize the energy consumption in a single Web tier has been done by Rajamani et al. [17]. They use a load distribution scheme to schedule the requests on a subset of servers and turn others off. However, our work is different from [17] in two ways. First, we propose a dynamic provisioning scheme to determine required number of servers in all the three tiers together. Second, we make use of DVFS and DPM for obtaining better energy savings.

Power/energy consumption in server farms has been looked at with different perspectives. Gandhi *et al.* [8] used a queuing theoretical model for allocating a given power budget across the servers so as to maximize performance. Chen *et al.* [3] made the first formalism for the problem of reducing server energy consumption in hosting centers running multiple applications, while meeting user specified performance bounds. They optimize energy efficiency by considering both static server provisioning and DVFS. However, our current work differs from theirs in three aspects. First, we consider dynamic server provisioning in a multi-tier data center environment unlike their evaluation for single tier of servers. Second, we optimize energy savings by considering all three techniques: dynamic provisioning, DVFS, and DPM. The work done in [3] does not consider the benefits of sleep states for servers in achieving further energy savings. Third, they decided CPU frequencies for servers by estimating the performance of a server farm against all the available CPU frequencies. We use a more efficient technique to find an energy optimal frequency at which a server selected by dynamic provisioning should run to minimize energy usage. Meisner *et al.* [13] proposed PowerNap to transition a system back and forth between a high performance state and a low power sleep state in response to instantaneous load to conserve power. The proposed heuristics in this work avoid a server's idle period by proactively putting a server to sleep much before the start of idle period, thereby, replacing the period of low server utilization with power saving sleep states. The authors in [9] proposed multi-mode energy management schemes for clusters running multi-tier applications. The authors make use of both DVFS and multiple sleep modes available in today's server systems to provide heuristics for power savings. Our work although has the same inspiration, but it differs from [9] in the following two ways. First, dynamic provisioning of servers is not considered in their work, which is one of the potential contribution of our work. Second, we explicitly address the impact of power management techniques on user perceived SLA and perform a detailed tradeoffs analysis between them using both simulations and a mathematical model.

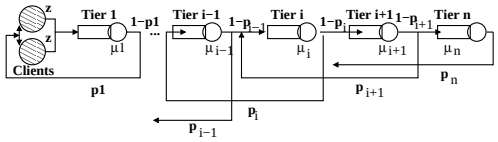


Fig. 1: A closed queueing network

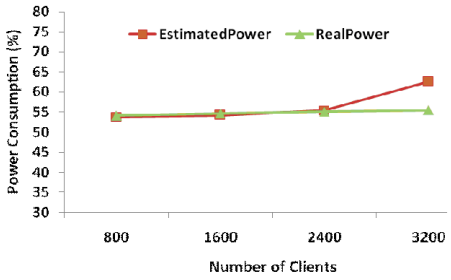


Fig. 2: Estimating power consumption with CPU utilization (results from the RUBiS benchmark)

III. Background

In this section, we describe system assumptions and our problem formulation along with definitions for subsequent sections.

We define an *active* server as one which is either in the turned-on or sleep state and an *inactive* server to be one in the turned-off state. We attempt to address the following two questions. First, given a total of N homogeneous servers in a multi-tier data center, how do we allocate $n \leq N$ active servers to process requests from users? Second, how do we schedule each active server's CPU speed s and decide on the parameters for DPM that minimize the energy consumption while satisfying the SLAs?

To model the performance aspects in our problem, we view a multi-tier data center as a closed queuing network (Figure 1) [23]. A user sends a new request to the front-end tier after waiting for a certain amount of time after getting the response for the previous request back. Requests served in a tier i is either forwarded to tier $i+1$ with a probability of $1-p_i$ or responded back to tier $i-1$ with a probability of p_i. We define the visit ratio to capture the phenomenon that a request received at a tier i can generate multiple requests to the next tier $i+1$, which commonly occurs between the Application server (AS) tier and the Database server (DB) tier. The servers in the same tier are assumed to have negligible intra-tier traffic [6]. The dynamic characteristics of the Internet traffic is captured by varying the number of users at the beginning of each time frame. Although a request consumes multiple resources in a server, the overall power consumption of a server can be estimated by the analysis of CPU utilization [7], [8]. In this work, the CPU utilization refers to the utilization of a server unless otherwise specified [7]. However, we consider all other resources in a server, especially for the sleep states.

We model the power consumption of a system as a function of its CPU utilization and speed, which follows the observation reported in [7], [8].

$$P(s,\rho) = \rho\left[a(s-s_{min})^{\alpha}+b\right], \alpha \geq 1\,, \tag{1}$$

where ρ is the utilization of a system, a is a coefficient which depends on the system, s_{min} is the lowest CPU speed, α is a factor that generalizes the CPU technology and b is the power consumption when the CPU is operated at s_{min}. Then, the peak power consumption is $P(s,1.0) = 1.0 \times [a(s-s_{min})^{\alpha}+b]$. As reported in [7], we assume that the power consumption of a system at a given CPU speed is linearly proportional to the CPU utilization of the system. Our assumption is further supported in [25] where the authors show the power consumption of a server is approximately linear to the server utilization which in turn can be estimated from CPU utilization. We also verified this by comparing actual power consumption from our prototype data center to the estimated power consumption based on CPU utilization as shown in Figure 2.

The energy consumption of a system is the integral of the consumed power over the measured time.

$$E = \int P(s,\rho)\,dt \tag{2}$$

We establish the relationship among the energy consumed for processing requests ($E_{dynamic}$) , idle period (E_{idle}), and sleep period (E_{sleep}) in the following corollary.

Corollary 1: For a given CPU speed s, CPU utilization ρ, time interval of length T, the ratio of idle power consumption to the peak power consumption $0 < k < 1$, and sleep power consumption to peak power consumption $0 < k' < 1$, the energy consumption E of a server is given by

$$E = P(s,1.0)\left[(T-t)(\rho(1-k))+k)+tk'\right]\,, \tag{3}$$

where t is the total amount of time when the server is in sleep state during the interval $[t_0, t_0+T]$.

Reasoning for the above corollary is given as follows. The energy consumption of an active server for a given time frame is the sum of the energy consumed when the system is in the dynamic, idle, and sleep states *i.e.,*

$$E = E_{dynamic} + E_{idle} + E_{sleep}$$

However, the right side of the previous equation is equal to

$$\int_{t_0}^{t_0+T} \{(\rho(1-k))P(s,1.0) + kP(s,1.0) + k'P(s,1.0)\}\,dt$$

Since we assume that the CPU speed remains constant for a certain time frame, the energy consumption is the product of elapsed time and consumed power according to Equation 3.

The energy consumption for a server is a function of the CPU speed and utilization. Thus, for a given CPU utilization, we can determine the energy-optimal CPU speed s_0 that minimizes the server energy consumption.

Corollary 2: Given the average number of instructions m required to process a request and the predicted number of requests r, the energy consumption E of a server is minimal at a certain speed s_0.

Proof: Let ρ be the utilization during T and $P(s,\rho)$ be the power consumption given in Equation 1. The energy

consumption E thus becomes

$$E = \int_{t_0}^{t_0+T} P(s,\rho)\,dt = \rho P(s,1.0)T \tag{4}$$

The total number of instructions required to process the requests during an interval T for a CPU speed s and CPU utilization ρ is $T \times s \times \rho$. A simple formula that relates the number of instructions during T to the average number of instructions required for a request is $Ts\rho = mr$. Therefore, the energy consumption E of a server is given by

$$E = P(s,1.0)mr/s \tag{5}$$

To find a CPU speed s_0 that minimizes E, we differentiate E w.r.t. CPU speed s.

$$\frac{d\,E}{d\,s} = \frac{mr(sP'(s,1.0) - P(s,1.0))}{s^2} \tag{6}$$

In $P(s,1.0) = a(s - s_{min})^{\alpha} + b$, if $\alpha = 1$, E is a strictly decreasing function since $\frac{d\,E}{d\,s} < 0$. Thus, the maximum CPU speed s_{max} minimizes the total energy consumption of a server, therefore s_0 = s_{max}. However, if $\alpha = 3$, there exists a $s_{min} \leq s_c \leq s_{max}$ such that $\frac{d\,E}{d\,s}(s_c) = 0$. Since CPU speeds are discrete, the CPU speed s_0 closest to s_c minimizes the energy consumption. When the idle and sleep state are considered, the proof remains similar. Hence, for processing a request, the energy consumption of a server is minimal at s_0. ■

Since the energy consumption is the product of processing time and consumed power, we should consider both of these factors for potential energy savings. Decreasing CPU speed lowers power consumption, but extends the processing time of a job. Increasing CPU speed shortens the processing time, but increases power consumption during that interval, which again may not save significant energy. However, when we cannot meet the performance goal with the energy optimal CPU speed, we have to run the CPU at the maximum possible speed to achieve the desired performance. In short, the CPU speed of a server should be greater than or equal to s_0 when we consider the trade-off between performance and energy.

IV. Problem formulation

The problem of optimizing energy consumption with performance constraint can be formulated as follows. Using Corollary 1, for a given average CPU speed s_i of servers in a given time frame T, the cost C (energy consumption per server) can be defined as

$$C = \begin{bmatrix} c_1 & c_2 & \cdots & c_n \end{bmatrix}, \tag{7}$$

where $c_i = P(s_i,1.0)[(T-t)(\rho_i(1-k)+k) + tk']$ and ρ_i is the utilization of a server in a tier $1 \leq i \leq n$. k and k' are hardware-dependent constants, which can be determined through real measurements. Variables in the cost function are s, ρ_i and t. Different values of the function $P(s_i,1.0)$ can be evaluated by measuring the power consumption of server at different speeds. The objective function representing the energy consumption of a multi-tier data center is given by

$$CV = \begin{bmatrix} c_1 & c_2 & \cdots & c_n \end{bmatrix} \begin{bmatrix} v_1 \\ v_2 \\ \vdots \\ v_n \end{bmatrix}, \tag{8}$$

where v_i in $V = \begin{bmatrix} v_1 & v_2 & v_3 & \cdots & v_n \end{bmatrix}^T$ is the number of active servers in tier i in a given time frame. Thus, our goal is to determine a vector V and the variables in the cost functions which will minimize the overall cost function (Equation 8) under the following constraints.

Suppose

- T : response time of a multi-tier data center
- T_{SLA} : response time constraint specified in SLA
- N : total number of servers
- τ_i : throughput of a server in tier $1 \leq i \leq n$
- r_i : expected number of requests for tier i per *time frame*

$$\begin{aligned} T &\leq T_{SLA} \\ v_i &\geq 1 \\ \sum_{i}^{n} v_i &\leq N \\ T_{SLA} \cdot \vec{\tau} \cdot V &\geq \begin{bmatrix} r_1 & r_2 & \cdots & r_n \end{bmatrix}^T, \end{aligned} \tag{9}$$

where $\vec{\tau}$ is a $n \times n$ matrix given by

$$\begin{bmatrix} \tau_1 & 0 & & \cdots & & 0 \\ 0 & \tau_2 & 0 & \cdots & & 0 \\ 0 & 0 & \tau_3 & 0 & \cdots & 0 \\ \vdots & & & \ddots & & 0 \\ 0 & & & \cdots & & \tau_n \end{bmatrix}$$

These constraints mean that the total response time should be bounded by the SLA, each tier has at least one active server, the sum of the active servers should be less than or equal to the total number of servers, and each tier should process the required number of requests within the target response time.

By Little's Law [12], the average number of requests in a system is equal to the product of throughput and the average response time of each request. Thus, the throughput of our system, $\vec{\tau} \cdot V$ multiplied by the target response time T_{SLA} must be larger than or equal to the expected number of requests to satisfy the performance bounds and save energy. For example, if we let r_i be 12 and τ_i 3, then we get $v_i = 4$. This means that 4 servers are sufficient to handle all 12 requests. However, if T_{SLA} is 2 seconds, we can reduce v_i to half (i.e. $v_i = 2$) and still manage to meet T_{SLA}. This is why we need to multiply T_{SLA} in the last constraint.

V. Methodology

In this section, we describe the methodologies used to solve the problem formulated in Section IV. Specifically, we describe how to estimate the model parameters and the heuristics used to solve the problem. The notations for the parameters used in our model along with the methods to obtain them are summarized in Table I.

TABLE I: Summary of model parameters

Parameters	Description	Methods to obtain
T_{SLA}	The maximal response time specified in SLA	Given
N	Total # of Servers	Given
ρ	CPU utilization	Actual Measurement
M	The # of requests	Actual Measurement
τ	Throughput of a server	Actual Measurement
T	Response time	Actual Measurement
$1/\mu$	Service Time of a request at current CPU speed	Estimated by ρ/τ
m	The Avg. # of instructions per requests)	Estimated in Section V-A
t_0	Service time at energy optimal CPU speed	Calibrated in Section V-A
$\vec{r}$	The # of requests for each tier in next time frame	Predicted by weighted moving average from actual measurements
$Power(s, \rho)$	Power consumption	Estimated by linear regression from actual measurements
τ_{max}	Throughput of a server to meet the response time	Estimated by MVA
$\bar{T}$	expected response time of incoming requests	Estimated by MVA
$\vec{v}$	# of servers for each tier in the next time frame	heuristic
s	CPU speed for processing incoming request	heuristic
t	The amount of time to sleep before processing incoming request	heuristic

A. Estimation of Model Parameters

The parameters of interest include peak power consumption, server utilization, number of requests in the next time frame and server throughput. Since the plate peak power consumption is far from the real peak power consumption [7], we use real *peak power consumption* in our model. The CPU *utilization* of a server is obtained from monitoring tools supported by the operating systems, for example, `sar` package in Linux. The *number of requests* received by a server in the next time frame is measured by observing the number of open sockets. A server opens a socket when a request arrives and keeps it open until the response gets back to the client as per the HTTP1.1 protocol specification. We determine the number of open sockets by analyzing operating system logs. We predict the number of requests in a tier for a given time frame as a weighted moving average of the number of requests during the previous time frames. It is to be noted that more accurate prediction techniques such as auto-regressive time series could also have been used. However, simple moving average serves as a good prediction technique in our experiments.

A server in a tier may not always operate at the energy optimal speed s_0. Therefore, we calibrate a server's service time to the service time at speed s_0. To calibrate the *service time* of a request when a CPU is operated at speed s_0, we first calculate the mean number of instructions requested for a request in a server at the current CPU speed s_c and then relate it to the target speed s_0. The mean number of insturctions required to process a request, m_i is obtained as follows. For a time period of duration t, CPU utilization ρ, average CPU speed $\bar{s}$, number of requests r_i, and number of servers v_i in a tier i during the previous time frame, m_i is given by

$$m_i = \frac{\bar{s}\rho t}{\frac{r_i}{|v_i|}} = \frac{\bar{s}\rho t|v_i|}{r_i}\,, \tag{10}$$

where ρ, $\bar{s}$, and r can be obtained from operating system supported tools. Now, let us derive a simple formula to relate m_i to the energy optimal speed s_0. When a CPU is operated at the speed s_0, the average service time t_0 of a request is given by

$$t_0 = \frac{m_i}{s_0}\,, \tag{11}$$

where m_i is the average number of CPU instructions for a request at tier i. Also, the total number of instructions for a request remains the same even though we vary CPU speeds. This calibration procedure aims at estimating the service time of a request in a server before changing the frequency in the dynamic provisioning algorithm.

TABLE II: Parameters used in Algorithm 1

Parameters	Descriptions
T_{SLA}	Target response time
M	# of requests from clients
t_{0_i}	Service time of tier i at CPU speed s_0
$\vec{r}$	# of incoming requests for each tier
z	Think time interval
t	Duration of time frame
$\vec{V}$	# of servers for each tier
L_i	# of queued requests for tier i
$\bar{T}$	Total average response time
$\bar{T}_i$	Estimated average response time of tier i
$\bar{\tau}_i$	Estimated throughput of tier i
N	Total # of servers

In our model, we assume that each tier is modeled as a queue in a closed queuing network. We use Mean Value Analysis (MVA) algorithm [18] to determine the maximal throughput and marginal response time of each tier. MVA takes as input number of clients, visit ratio of each tier, number of tiers, and user think time. The parameters obtained from MVA algorithm determine the elements of $\vec{\tau}$ in Equation 9. The elements of vector V are all integers, leading to the above problem formulation as an integer linear programming model which is known to be NP-complete. Hence, we propose heuristics to solve the above problem, though they may not always guarantee an optimal solution. In the next section, we describe our heuristics.

B. Heuristics

We propose two heuristics to solve our proposed optimization model. The first heuristic involves a global optimization for finding the desired number of active servers in each tier for reducing the energy consumption by using just sufficient number of servers. The second heuristic relates to a local optimization at individual server levels. It determines the energy optimal CPU speeds and sleep states.

The first heuristic algorithm (Algorithm 1) works as follows: We get the estimated throughput, response time, and queue length for each tier from the MVA algorithm. As described in

Algorithm 1 Heuristic for selecting the optimal number of servers for each tier of a multi-tier data center

Input: T_{SLA}, M, $\vec{r}$, s_0, $\bar{s}$, ρ, $\vec{V}_{cur}$, t, z, k, n
1: $m_i = \frac{\bar{s}\rho_i t|v_i|}{r_i}$
2: $t_{0_i} = \frac{m_i}{s_0}$
3: $(\bar{T}, \bar{T}_i, \bar{\tau}_i) = MVA(M, t_{0_i}, \vec{r}, z, T_{SLA}, n)$
4: $L_i = \tau_i \times T_i$
5: $v_i = \frac{r_i + L_i}{\bar{\tau}_i T_{SLA}}$
6: **if** $v_i > v_{i_{cur}}$ **then**
7: $v_i = v_{i_{cur}} + 0.9|v_i - v_{i_{cur}}|$
8: **else if** $v_i < v_{i_{cur}}$ **then**
9: $v_i = v_{i_{cur}} - 0.2|v_i - v_{i_{cur}}|$
10: **end if**
11: **if** $\sum_{i=1}^{n} v_i \geq N$ **then**
12: $v_i = \frac{N v_i}{\sum_{i=1}^{n} v_i}$
13: **end if**

Algorithm 2 CPU speed scheduling of a server in tier i

Input: T_{SLA}, $\bar{T}_i$, $\bar{\tau}_i$, N, r, s_0, ρ, t, $\bar{s}$
1: $m = \frac{\bar{s}\rho t}{r}$
2: $L_i = \bar{T}_i \times \bar{\tau}_i$
3: **if** $\frac{(L_i+1)m}{\bar{T}_i} < s_0$ **then**
4: sleep for $(\bar{T}_i - (L_i + 1)m/s_0)$
5: run the CPU at s_0
6: **else**
7: **if** $(L_i + 1)m/\bar{T}_i < s_{max}$ **then**
8: run the CPU at the speed closest to $\lceil (L_i + 1)m/\bar{T}_i \rceil$
9: **else**
10: run the CPU at s_{max}
11: **end if**
12: **end if**

section IV, the number of servers v_i for tier i is obtained by

$$v_i = \frac{L_i + r_i}{T_{SLA} \times \bar{\tau}_i} \tag{12}$$

The performance of this algorithm might be sensitive to the time interval between provisioning. However, the algorithm itself does not need to specify the time interval.

The intuition behind this heuristic is that we can save energy with agreeable performance degradation by minimizing the number of servers. Since the throughput τ_i is the maximum possible throughput that can satisfy the target response time, the number of servers v_i, given by our heuristic is the minimum number of servers that can meet the given SLA and achieve substantial power/energy savings. Note that our method does not search the entire solution space to find the appropriate number of servers. When the number of active servers required is greater than N, we allocate all of these N servers to process the given workload. We use the simple Round-Robin scheme to distribute workload across the servers. Also, to prevent frequent fluctuation of the number of servers in each tier, we gradually change the number of servers as described in lines 6 - 9 of Algorithm 1.

Next, Algorithm 2 describes the heuristic for dynamically determining the duration of sleep states and CPU speed to account for the dynamic nature of the Internet traffic. This problem is considered NP-hard [10]. Our approach may put a CPU into the sleep state while delaying the incoming and already queued requests, provided that we can process the delayed requests at an energy optimal CPU speed s_0 without violating the given time limit. However, the scheme presented in [13] makes a system sleep whenever it detects an idle state and then wakes up on the arrival of the next job. Our approach is motivated by the fact that a server sleep power consumption is around 10% of peak power consumption [13].

In order to find the duration for the sleep period, we first estimate the response time of an incoming request. Since an incoming request will be processed after all other queued requests are completed in a First-Come-First-Serve discipline and the service rate of a CPU is assumed to be the CPU speed, the response time of an incoming request is given by

$$\frac{(L_i + 1)m}{s},$$

where L_i is the number of requests in the queue, m is the average number of instructions required to process a request, and s is the CPU speed. To minimize the energy consumption, we need to run the server at a speed $s = s_0$ as given in Corollary 2. Thus, the current incoming request will be completed at $(L_i + 1)m/s_0$.

If we can finish processing the current incoming request within the marginal response time of a request in tier i, $(L_i + 1)m/s_0 < \bar{T}_i$, we can put the system into the sleep state while keeping the response time within T_{SLA}. When a request arrives, we inspect the expected response time of the incoming request as above. If $\bar{T}_i - (L_i + 1)m/s_0 > 0$, we can transit the server into the sleep state for $\bar{T}_i - (L_i + 1)m/s_0$ duration of time. After that duration, the server wakes up and runs at the energy optimal speed. If the incoming request cannot be completed within $\bar{T}_i$, we run the CPU at least at the energy optimal speed without transitioning into the sleep state. Since the overhead of transition from/to the sleep state in terms of performance is negligible [13], the above approach does not hurt the response time, specified in SLA.

As shown in Corollary 2, we do not need to run the system under the energy optimal CPU speed s_0 even when we can process the incoming request with a slower CPU speed than s_0. However, when the given workload cannot be processed within $\bar{T}_i$ at the speed s_0, we need to run the CPU at a speed s, where $s_0 < s \leq s_{max}$. The speed s is the closest available CPU speed to $(L_i + 1)m/\bar{T}_i$. When $s > s_{max}$, we need more servers to process the current workload and the number of servers will be increased in the next time frame as explained in Algorithm 1.

Our heuristic for selecting the minimal number of servers picks the servers for each tier without searching the solution space exhaustively. A single tier data center environment does not take into consideration the chain reaction that may occur in a multi-tier environment when the number of servers in a tier is changed. Unlike a single tier case, in a multi-tier data center, one has to consider the behavior of other tiers, while selecting servers for a tier. Also, both DVFS and DPM should be considered together based on the estimated results from an entire tier granularity and not from individual server in a tier. This leads to the optimal allocation of servers across all the tiers as well as avoids the global performance degradation caused by local optimization technique such as DVFS and DPM done at the granularity of a server.

C. Intuitions behind possible energy savings

Next, we analyze the trade-off between performance and possible energy savings with our proposed heuristics. For the predicted number of requests for next time frame r, the number of servers in tier i v_i is equal to $(r_i + L)/\bar{\tau}_i T_{SLA}$ in our heuristics (in Algorithm 1). That is, we use enough number of servers to meet the target response time as explained in section V-B. However, our heuristics for the CPU speed considers the energy optimal CPU speed rather than the maximal CPU speed as given in Corollary 2. The performance degradation of our proposed scheme is graceful since we can adjust the speed dynamically to avoid violating the SLA.

Let us now discuss the possible energy savings that can be obtained with our schemes. According to Equations 8 and 9 in section IV, the total energy consumption of a data center increases linearly with the increase in number of active servers. From experimental results, it was confirmed that the number of servers provisioned by our scheme is almost similar to that of dynamic provisioning with maximal CPU speed. Thus, when the traffic fluctuates, we can achieve energy savings by reducing the number of active servers under light traffic.

Towards this end, the individual servers also utilize DPM which further reduces their energy consumption. The system utilization of a server with our scheme is at least

$$\rho = \tau \times \text{Service Time} = \frac{L}{T} \times \frac{m}{s_0}, \tag{13}$$

where L is the average queue length, T is the target response time, m is the average number of instructions per request, and s_0 is the energy optimal CPU speed. Since, we put the system including CPU into the sleep state when

$$T_i s_0 > (L_i + 1)m$$

(line 3 of Algorithm 2), we can yield the lowest utilization of servers with our scheme as follows.

$$\begin{aligned} \rho &= \tau \times \text{Service Time} && \text{(Utilization Law)} \\ &= \frac{(L_i + 1)}{T_i} \times \frac{m}{s_0} && \text{(Little's Law)} \end{aligned}$$

Thus, we replace the period of idle state as well as low utilization with the sleep state which consumes 10 % of peak power consumption as reported in [13].

Finally, since the power consumption is a cubic function of the CPU speed, if we operate a server with the energy optimal speed rather than maximal or minimal speed, we achieve more energy savings. For example, as suggested in [8], suppose the relation between power consumption and CPU speed is given by

$$P(s) = \left(\frac{100}{39}\right)^3 (s - 1.2)^3 + 150\,, \quad 1.2 \le s \le 3.0\,,$$

The energy consumption of a server is minimized for s = 2.8 GHz according to Corollary 2. We assume that there is no job dependency with other machines. Using the above power-to-frequency equation, the power consumption for $s = 3.0GHz$ is 250W and for $s = 2.8GHz$ is 219W. Therefore, the ratio of energy consumption to process unit cycle for $s = 2.8GHz$ to $s = 3.0GHz$ is

$$219W/2.8GHz : 250W/3.0GHz = 0.938 : 1.0$$

Thus, by only using the energy optimal speed, we could reduce the power consumption of each machine by 6.2 %, compared to running them at maximal CPU speed.

To summarize, our proposed schemes can achieve substantial energy/power savings by reducing the number of servers required, using energy optimal speed instead of running servers at either the maximal CPU speed or any randomly picked CPU speed, and utilizing dynamic power management with the sleep state. We obtain potential energy savings, while satisfying SLAs.

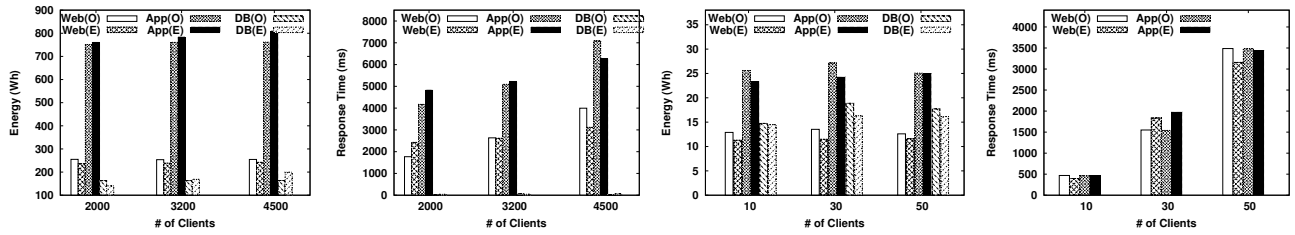
VI. Experimental Results

A. Validation of the Simulator

We validated the correctness of our simulator from real measurements on a prototype 25 servers three-tier data center with two multi-tier application benchmarks. For validation, we compared the response time and energy consumption obtained from simulation with those collected from real measurements. Baseline technique for comparison with our schemes uses static provisioning without use of either DVFS or DPM. For simulation, we used a modified version of a multi-tier data center simulator [11], which is written in CSIM [14]. The parameters used in the simulation like service time, peak power consumption, duration of time frame, etc are all derived from real measurements (Table I).

We performed experiments in two different hosting environments. The first one, END1 is a traditional multi-tier data center and the second one, ENV2 is a virtualized multi-tier data center. The first hosting environment includes machines with dual 64-bit AMD Opteron 250 CPUs (2.39GHz) and 4GB main memory. The operating system is Red Hat Enterprise Linux 4.0 (Linux kernel v2.6.9). Apache 2.0.54 is used for web servers, JOnAS 4.6.6 for application servers, and MySQL 5.0.15-0-Max for database servers. Servers are connected by 1Gbps Ethernet. Communication between web and application servers is made through the mod_jk v.1.2.5 connector and that between application and database servers is via the C-JDBC v.2.0.1 connector. For the workload, we used the RUBiS benchmark [4] which models an online auction site.

The second experimental environment (ENV2) consists of four Xen-based virtual machines [1] that run on four different physical machines. Each physical host machine is equipped with dual Intel Xeon CPUs (3.4GHz) and 2GB RAM, running on Redhat 9 Enterprise edition with kernel version of 2.6.18-164.el5. We installed Xen v3.1.4 on all of these four machines and created one Virtual Machine (VM) on each physical machine. Each VM runs on Fedora core 4 having kernel version 2.6.18.8. VMs are given 512MB RAM and connected by 1Gbps Ethernet. The four VMs are distributed across the three tiers in the following manner - first tier consists of an Apache web server running on one VM, two JBoss 3.2.8SP1 application servers running on separate VMs comprise the second tier, and MySQL 4.1 database server



(a) Energy Consumption (ENV1) (b) Response Time (ENV1) (c) Energy Consumption (ENV2) (d) Response Time (ENV2)

Fig. 3: Validation results of the model and simulator

hosted on another VM forms the third tier. We used TPC-W benchmark that models a three-tier online book store and the J2EE implementation of TPC-W [21] was used. For client emulator, we used a TPC-W [20] based workload generator. Yokogawa's WT210 power meter was used to record the power consumption of machines.

Figure 3 shows the validation results with respect to energy consumption and response time. Here, (O) represents the observed results from real measurements and (E) represents the estimated results from our simulator. (The energy numbers represent the total energy consumption across the cluster). From these figures, we observe that the graphs for the response time and energy consumption corresponding to real measurements and simulation results closely match with each other for different number of clients in both RUBiS and TPC-W benchmarks. We also notice that the height of the bar graph for DB tier in Figure 3b is negligible due to the small response time of DB tier. Moreover, since the version of MySQL that we used does not provide response time information, we were unable to validate response time of the DB tier in Figure 3d for TPC-W. (We, however, expect similar trend for the DB tier as was observed for the WS and AS tiers.) However, it was possible in ENV1 because we could modify the RUBiS benchmark to log DB tier response time.

Since, response time and energy consumption are well captured by our simulator, according to Corollary 1, it implies that CPU utilization and, hence, throughput can also be correctly estimated using our simulator. We use our multi-tier data center simulation platform for detailed analysis of the proposed schemes. We obtained results from our simulator instead from real measurements because of the unavailability of sufficient number of servers in our experimental cluster.

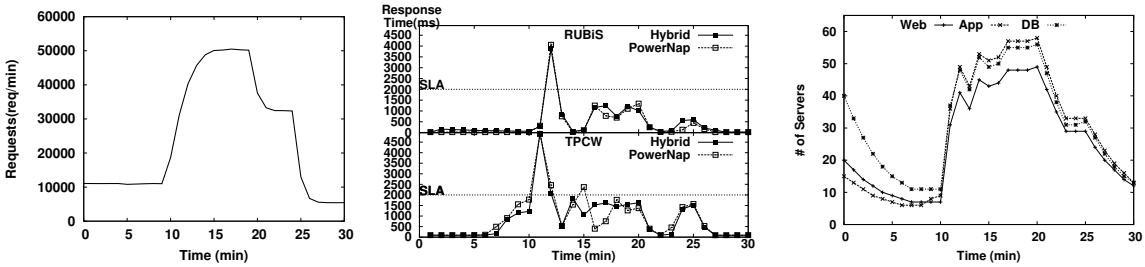
B. Evaluation Results

In this section, we present the simulation results to compare our heuristics, Hybrid, with four other schemes, STATIC, DYN, DVFS and PowerNap. The baseline is the static provisioning at maximal CPU speed without invoking DVFS pr DPM. For the static provisioning, we found the optimal number of servers for each tier using the MVA algorithm to guarantee an average response time of 2 seconds (which is the SLA response time limit in our experiments). The dynamic provisioning (DYN) decides the number of servers for each tier based on heuristic 1 and then the CPUs run at the maximal speed. DVFS is the scheme that conflates heuristic 1 and heuristic 2 without using DPM. PowerNap employs heuristic 1 with the available DPM mechanism [13].

a) Experimental Details: We varied the CPU speed from 1.2 GHz to 3.0 GHz and estimated power consumption of a node using the equation in [8]. In our simulations, servers are modeled as M/M/1 queues for simplicity. From real measurements with the RUBiS and TPC-W benchmarks, we obtained various simulation parameters like service time, ratio of dynamic to static requests and number of database queries generated per dynamic request. We varied the number of requests for the two benchmarks dynamically over a duration of 30 minutes and the request variation with time is shown in Figure 4a. In our experiments, the dynamic provisioning scheme decides the number of servers for each tier during the next time at the granularity of one minute interval.

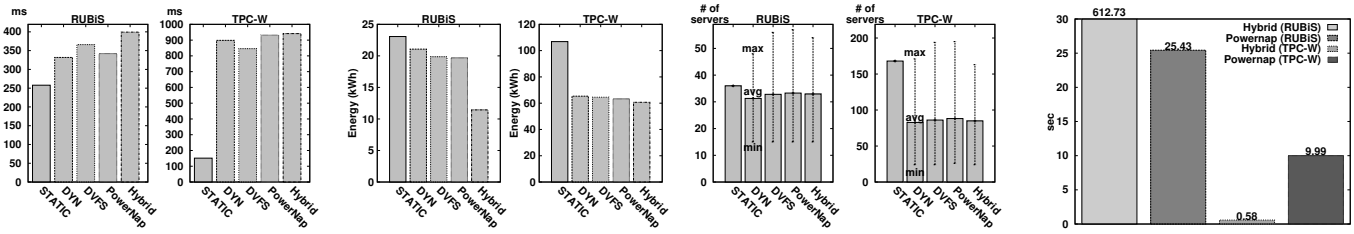
b) Simulation Results: Figure 4b shows the variation of response time of Hybrid and PowerNap for the RUBiS and TPC-W workload for each time frame. The number of servers provisioned by our heuristic across all the tiers in a multi-tier data center is shown in Figure 4c. We observe that except the spike in these graphs during the 10-12 minute time interval (corresponds to flash crowd), Hybrid keep the response time below 2000*ms* and thus adheres to the response time bounds of SLA. Although the response time of our scheme hikes up to 5000 *ms* during the flash crowd, within the next couple of intervals, our scheme could allocate enough number of servers across the tiers to keep the response time within the SLA limit of 2000ms. The PowerNap scheme showed similar trend. The results demonstrated that the proposed scheme, Hybrid can adapt to both static and burst Internet traffic. Although not shown in these graphs, other schemes also exhibited similar behavior for both RUBiS and TPC-W.

Figure 5 provides comparative analyses of the average response time, energy consumption, the number of of powered-on servers, and the total sleep time per host. Figure 5a represents the average response time of all requests for the RUBiS and TPC-W workloads, respectively. As expected, static has the minimum latency due to overprovisioning and Hybrid incurs the maximum latency since Hybrid can make a system sleep longer than PowerNap. For the TPC-W workload, since one dynamic request generates around 55 database queries, leading to the presence of very small requests, the system has too small idle periods to be utilized for DPM. Thus, the sleep state cannot be utilized as much as for the RUBiS workload, which makes the average response times of all dynamic provisioning schemes similar to each other. We note that the response time of DVFS is 7% higher than PowerNap for the RUBiS workload because the servers in DVFS process the requests at a lower speed than PowerNap. For the TPC-W



(a) Workload (b) Response time (c) # of servers of each tier for TPC-W

Fig. 4: The effect of dynamic provisioning in Algorithm 1



(a) Average Response Time (b) Energy Consumption (c) The number of powered-on servers (d) Total sleep duration per host

Fig. 5: Evaluation of the proposed method

workload, however, the response time of DVFS is lower than other dynamic schemes.

Figure 5b shows that Hybrid can reduce the total energy consumption by about 50% and 44% for RUBiS and TPC-W, respectively, when compared to STATIC. Hybrid could save 42% more energy compared to PowerNap for the RUBiS workload. The energy saving is attributed to the longer sleep period and energy optimal CPU speed in Hybrid than PowerNap, which is described in Section V. As described in Section V, we observe that the total sleep period of a host with Hybrid is 24 times longer than that of PowerNap for the RUBiS workload (Figure 5d). The average duration of one sleep period with Hybrid is 18.66 *ms*, compared to 0.48 *ms* with PowerNap. Hybrid resulted in 32836 transitions to the sleep state per host on average, while PowerNap had 52512 transitions on average. With PowerNap, the system wakes up whenever a request arrives, creating longer low utilized periods in the system than with Hybrid. Also, we took the advantage of DVFS by running the active servers at the energy optimal speed instead of at the maximal speed. The energy saved by PowerNap, DVFS, and DYN, compared to static provisioning for the RUBiS workload is about 15%, 14%, and 10%, respectively.

Since the behavior of a system varies according to the workload, the results from the TPC-W workload is different from that of RUBiS. In Figure 5b, the energy consumption for TPC-W workload for all dynamic provisioning schemes is reduced by more than 49% relative to STATIC. Since the number of servers to process requests during the peak traffic period is significantly larger, compared to the case with low traffic, dynamic provisioning schemes can save significant energy. However, because of many small database queries in TPC-W workload, the idle period between requests is short. Thus, the opportunity for utilizing the sleep state is less for the TPC-W workload, although the overall energy consumption decreases compared to the STATIC provisioning.

The number of active servers for both the workloads is shown in Figure 5c. In static provisioning, a total of 36 and 168 servers were used during all time frames for the RUBiS and TPC-W workloads, respectively. For the RUBiS workload, the number of active servers in STATIC is close to the average number of active servers in all dynamic schemes. However, for the TPC-W workload, the number of servers with STATIC is close to the maximal number of active servers in dynamic schemes due to high variability in number of requests generated between tiers. In all the dynamic provisioning schemes, the number of active servers for all the tiers decreased under light traffic, which corresponds to the minimum number of active servers (a total of 17 servers across all the tiers in Figure 5c). However, we used a total of 36 servers in STATIC. 168 servers were provisioned with STATIC for the TPC-W workload, but only 25 servers were allocated with all dynamic provisioning approaches. This is because STATIC reserves adequate number of servers for handling peak traffic even under low load conditions. At the eleventh time frame (Figure 4a), the dynamic provisioning schemes increase the number of servers to maintain the response time of requests due to the presence of flash crowd in the previous time frame. During heavy traffic periods, for the RUBiS workload, PowerNap and Hybrid provisioned 16% and 12% more servers than DYN, respectively. However, for the TPC-W workload, the number of active servers in PowerNap is 13% more than DYN, while that of Hybrid is 5% less than DYN. This implies that the server utilization in Hybrid is higher than that of PowerNap under heavy traffic. To summarize, our method creates more opportunities for energy savings, while maintaining the SLA since it incorporates dynamic provisioning with the energy

management techniques for individual servers.

Some insights drawn from the experimental results are summarized below. First, our proposed scheme, Hybrid, may have less potential to save energy where the DPM technique cannot be leveraged much; for example, in the TPC-W workload, which generates many small requests between tiers. Second, experimenting with different response time thresholds, we observe that the behavior of all the considered dynamic schemes does not significantly change. With a stricter SLA constraint, the overall energy consumption increases by allocating more number of servers to keep up with the performance bound in dynamic schemes. Similarly, with a relaxed SLA constraint, the energy usage reduces since dynamic schemes can achieve the desired performance with less number of servers. We experimented with 1000 and 3000 ms SLA thresholds, but the results are not shown due to space constraints.

VII. Conclusions

In this paper, we proposed a novel multi-level approach for increasing the energy efficiency and maintaining the performance bounds of a multi-tier data center. We formulated our performance and energy analysis as an optimization problem and introduced two heuristics based on queuing theory to solve it. The first heuristic dynamically provisions sufficient number of servers across each tier to conserve energy, while satisfying the SLAs. The second heuristic schedules the above selected servers to run at the energy optimal CPU speed and utilizes the DPM mechanism involving sleep states to boost further energy savings. We have constructed a prototype three-tier data center to both validate our simulator as well as obtain parameters for solving our proposed model. Experimental results with the RUBiS and TPC-W benchmarks demonstrated that our proposed Hybrid technique can provide up to 50% more energy savings compared to the base scheme of static provisioning where the servers run at the maximal speed without DPM. Extensive simulation results demonstrated that the combined Hybrid approach to energy saving is more effective then other proposed dynamic techniques such as DYN, DVFS and PowerNap, for workloads that provide opportunity to exploit the local DVFS and DPM techniques at individual servers. For the RUBiS workload, our integrated approach provided additional 42% energy savings and turned on less number of servers under heavy traffic compared to the most recently proposed PowerNap technique. The proposed approach delivers higher energy efficiency by facilitating longer sleep duration and/or reduced number of active servers.

For our future work, we plan to extend our scheme to environments with dynamic workloads consisting of multiple types of applications, heterogeneous servers and shared resources among servers.

References

[1] Paul Barham, Boris Dragovic, Keir Fraser, Steven Hand, im Harris, Alex Ho, Rolf Neugebauer, Ian Pratt, and Andrew Warfield. Xen and the art of virtualization. In *SOSP '03: Proceedings of the 19th ACM Symposium on Operating Systems Principles*, 2003.

[2] Jeffrey S. Chase, Darrell C. Anderson, Prachi N. Thakar, Amin M. Vahdat, and Ronald P. Doyle. Managing energy and server resources in hosting centers. In *SOSP '01: Proceedings of the eighteenth ACM symposium on Operating systems principles*, 2001.

[3] Yiyu Chen, Amitayu Das, Wubi Qin, Anand Sivasubramaniam, Qian Wang, and Natarajan Gautam. Managing server energy and operational costs in hosting centers. *SIGMETRICS Perform. Eval. Rev.*, 33(1):303–314, 2005.

[4] OW2 consortium. Rubis:rice university bidding system. http://rubis.ow2.org.

[5] E.N. Elnozahy, Michael Kistler, and Ramakrishnan Rajamony. Energy-efficient server clusters. In *Proceedings of the 2nd Workshop on Power-Aware Computing Systems*, 2002.

[6] Deniz Ersoz, Mazin S. Yousif, and Chita R. Das. Characterizing network traffic in a cluster-based, multi-tier data center. In *ICDCS '07: Proceedings of the 27th International Conference on Distributed Computing Systems*, 2007.

[7] Xiaobo Fan, Wolf-Dietrich Weber, and Luiz Andre Barrose. Power provisioning for a warehouse-sized computer. In *ISCA '07: Proceedings of the ACM international Symposium on Computer Architecture*, 2007.

[8] Anshul Gandhi, Mor Harchol-Balter, Rajarshi Das, and Charlse Lefurgy. Optimal power allocation in server farms. In *SIGMETRICS '09: Proceedings of the eleventh international joint conference on Measurement and modeling of computer systems*, 2009.

[9] Tibor Horvath and Kevin Skadron. Multi-mode energy management for multi-tier server clusters. In *PACT '08: Proceedings of the 17th international conference on Parallel architectures and compilation techniques*, pages 270–279, New York, NY, USA, 2008. ACM.

[10] Sandy Irani, Sandeep Shukla, and Rajesh Gupta. Algorithms for power savings. *ACM Trans. Algorithms*, 3(4):41, 2007.

[11] Seung-Hwan Lim, Bikash Sharma, Gunwoo Nam, Eun Kyoung Kim, and Chita R. Das. Mdcsim:a multi-tier data center simulation platform. In *IEEE Cluster Computing*, 2009.

[12] J.D.C. Little. A proof of the queueing formula:L=λW. *Operations Research*, 9:383–387.

[13] David Meisner, Brian T. Gold, and Thomas F. Wenisch. PowerNap: eliminating server idle power. In *ASPLOS '09: Proceeding of the 14th international conference on Architectural support*, 2009.

[14] Mesquite Software. CSIM. http://www.mesquite.com.

[15] Eduardo Pinheiro, Ricardo Bianchini, Enrique V. Carrera, and Taliver HEath. Load balancing and unbalancing for power and performance in cluster-based systems. In *Workshop on Compilers and Operating Systems for Low Power*, 2001.

[16] Ramya Raghavendra, Parthasarathy Ranganathan, Vanish Talwar, Zhikui Wang, and Xiaoyun Zhu. No "power" struggles: coordinated multi-level power management for the data center. *SIGARCH Comput. Archit. News*, 36(1):48–59, 2008.

[17] K. Rajamani and C. Lefurgy. On evaluating request-distribution schemes for saving energy in server clusters. In *ISPASS '03: Proceedings of the 2003 IEEE International Symposium on Performance Analysis of Systems and Software*, 2003.

[18] M. Reiser and S. S Lavenberg. Mean-value analysis of closed multichain queuing networks. *J. ACM*, 27(2), 1980.

[19] Vivek Sharma, Arun Thomas, Tarek Abdelzaher, Kevin Skadron, and Zhijian Lu. Power-aware qos management in web servers. In *RTSS '03: Proceedings of the 24th IEEE International Real-Time Systems Symposium*, 2003.

[20] TPC-W: Benchmarking An E-Commerce Solution. http://www.tpc.org/information/other/techarticles.asp.

[21] NYU-TPC-W. http://www.cs.nyu.edu/pdsg/.

[22] B. Urgaonkar, P. Shenoy, A. Chandra, and P. Goyal. Dynamic provisioning of multi-tier internet applications. In *Autonomic Computing, 2005. ICAC 2005. Proceedings. Second International Conference on*, 2005.

[23] Bhuvan Urgaonkar, Giovanni Pacifici, Prashant Shenoy, Mike Spreitzer, and Asser Tantawi. An analytical model for multi-tier internet services and its applications. In *SIGMETRICS '05: Proceedings of the 2005 ACM SIGMETRICS international conference on Measurement and modeling of computer systems*, 2005.

[24] ENERGY STAR Program U.S. Environmental Protection Agency. EPA report to congress on server and data center energy efficiency, Aug 2007.

[25] Zhikui Wang, Cliff McCarthy, Xiaoyun Zhu, Partha Ranganathan, and Vanish Talwar. Feedback control algorithms for power management of servers. In *3rd Workshop on Feedback Control Implementation and Design in Computing Systems and Networks (FeBID'08)*, Jun 2008.