

Managing Server Energy and Operational Costs in Hosting Centers

Yiyu Chen
Dept. of IE
Penn State University
University Park, PA 16802
yzc107@psu.edu

Amitayu Das
Dept. of CSE
Penn State University
University Park, PA 16802
adas@cse.psu.edu

Wubi Qin
Dept. of ME
Penn State University
University Park, PA 16802
wqin@psu.edu

Anand Sivasubramaniam*
Dept. of CSE
Penn State University
University Park, PA 16802
anand@cse.psu.edu

Qian Wang
Dept. of ME
Penn State University
University Park, PA 16802
quw6@psu.edu

Natarajan Gautam
Dept. of IE
Penn State University
University Park, PA 16802
ngautam@psu.edu

ABSTRACT

The growing cost of tuning and managing computer systems is leading to out-sourcing of commercial services to hosting centers. These centers provision thousands of dense servers within a relatively small real-estate in order to host the applications/services of different customers who may have been assured by a service-level agreement (SLA). Power consumption of these servers is becoming a serious concern in the design and operation of the hosting centers. The effects of high power consumption manifest not only in the costs spent in designing effective cooling systems to ward off the generated heat, but in the cost of electricity consumption itself. It is crucial to deploy power management strategies in these hosting centers to lower these costs towards enhancing profitability. At the same time, techniques for power management that include shutting down these servers and/or modulating their operational speed, can impact the ability of the hosting center to meet SLAs. In addition, repeated on-off cycles can increase the wear-and-tear of server components, incurring costs for their procurement and replacement. This paper presents a formalism to this problem, and proposes three new online solution strategies based on steady state queuing analysis, feedback control theory, and a hybrid mechanism borrowing ideas from these two. Using real web server traces, we show that these solutions are more adaptive to workload behavior when performing server provisioning and speed control than earlier heuristics towards minimizing operational costs while meeting the SLAs.

Categories and Subject Descriptors: I.6.5 [Model Development]: Modeling methodologies; D.2.8 [Metrics]: Performance measures

*Contact author

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.

SIGMETRICS'05, June 6–10, 2005, Banff, Alberta, Canada.
Copyright 2005 ACM 1-59593-022-1/05/0006 ...\$5.00.

General Terms: Design, Performance

Keywords: Energy Management, Performance Modeling, Feedback Control, Server Provisioning

1. INTRODUCTION

The motivation for this paper stems from two growingly important trends. On the one hand, the cost and complexity of system tuning and management is leading numerous enterprises to offload their IT demands to hosting/data centers. These hosting centers are thus making a considerable investment in procuring and operating servers to take on these demanding loads. The other trend is the growing importance of energy/power consumption of these servers at the hosting centers, in terms of the electricity cost to keep them powered on, as well as in the design of extensive cooling systems to keep their operating temperatures within thermal stability limits for server components. A careful balance is needed at these centers to provision the right number of resources to the right service/application being hosted, at the right time, in order to reduce their operational cost while still meeting any performance based service level agreement (SLA) decided upon earlier. Focusing specifically on the energy consumption problem, this paper presents three main techniques - a pro-active one, a reactive one, and a hybrid between the two - to dynamically optimize operating costs while meeting performance-based SLAs.

The growth of network-based commercial services, together with off-loading of IT services, is leading to the growth of hosting/data centers that need to house several applications/services. These centers provision thousands of servers, and data storage devices, to run these applications, earning revenue from these application providers (customers) in return - sometimes referred to as on-demand computing. Over-provisioning the service capacity to allow for worst case load conditions is economically not very attractive. Consequently, much of prior work (e.g. [9, 31, 36]) has looked at finding the right capacity, and distributing this capacity between the different applications based on their SLAs. However, there could still be time periods during the execution when the overall server capacity is much higher than the current demands across the applications. It should be noted that the hosting center is still incurring operational costs, such as electricity, during such periods.

Energy consumption of these hosting/data servers is becoming a serious concern. As several recent studies have pointed out [11, 12, 22, 23, 29] data centers can consume several Megawatt. It is not just the cost of powering these servers, but we need to also include the cost of deploying cooling systems (which in turn consume power) to ensure stable operation [27]. We are at power densities of around 100 Watts per square feet, and the cooling problem is expected to get worse with shrinking form factors. Finally, one also needs to be concerned with the environmental issues when generating and delivering such high electric power capacities.

Statically provisioning the number of servers for a given cost (say of electricity) and performance SLA can be conservative or miss out on opportunities for savings, since workloads are typically varying over time. Dynamic power management is thus very important when deploying server farms/clusters, and hardware has started providing assistance to achieve this. For instance, in addition to powering down a server, most processors today allow dynamic voltage/frequency scaling (DVS), where the frequency (and voltage) can be lowered to produce much more savings in power consumption compared to how much one loses on performance. While this may not provide as much savings as shutting down a server completely, the advantage is that it can still service requests (albeit at a lower frequency), and does not incur as high costs (whether it be time for transitioning between frequencies or in terms of wear-and-tear associated with repeated server on-off cycles).

Much of the earlier work [11, 29] in this area, used server turn off/on mechanisms for power management. The only other work that considered a combination of these two mechanisms (turn off and DVS) was done at IBM [15]. However, this work did not (nor did the others) consider the impact of server off/on cycles on the long term reliability of server components due to wear-and-tear. Note that failing of components incurs additional costs (hardware and personnel) for procurement and replacement. Further, none of the prior studies have really considered the goal of meeting a response time requirement (SLA). Rather, they have tried to optimize energy first, at a slight degradation in response time. For a hosting center, it is more important to meet the required SLA in terms of response time (since that is the revenue maker), and reducing the energy consumption when meeting this goal should be a desirable (rather than vice-versa). As our results will show, not treating the performance-based SLA in the dynamic optimization as a first class citizen causes these other schemes to fall short, even though they may provide more energy savings. Finally, most of the prior work has just looked at optimizing the energy in a single application setting, assuming the existence of a higher level server provisioning mechanism across applications. Our framework on the other hand, integrates server provisioning across applications (which automatically tells us how many servers to turn on/off) with energy management within an application, in an elegant fashion while still allowing different options for these two steps.

This paper presents the first formalization to this dynamic optimization problem of server provisioning and DVS control for multiple applications, which includes a response-time SLA, and the costs (both power and wear-and-tear) of server shutdowns. We present three new online mechanisms to this problem. The first is pro-active in that it predicts workload behavior for the near future, and uses this information in a stochastic queuing model to perform control. The second is reactive in that it uses feedback control to achieve the same goals. The third is a hybrid scheme where we use predictive information from the first for server provisioning, and feedback control of the latter for DVS. Using real web-server workloads in a multi-application setting, we show that all these schemes can provide good energy savings without ignor-

ing the SLA (unlike the only other related mechanism [15] we can compare – it is never able to meet the SLA).

2. SYSTEM MODEL

Our hosting center model contains a number of identical servers, called a server cluster, which are all equally capable of running any application. While most of our system models and optimization mechanisms are applicable to a diverse spectrum of application domains (from those in the scientific domain to those in the commercial world), the implementation and evaluations are mainly targeted towards server based applications (web servers in particular) that service client requests. Each HTTP request is directed to one of the servers running the application, which processes the request in a time (service time) that is related to a request parameter (e.g. file size). Consequently, one can impact the throughput/response time for these requests by modulating the number of servers allocated to an application at any time.

Dense blade systems are being increasingly deployed in hosting centers because of their smaller form factors (making it easier to rack them up), and power consumption characteristics. Such systems provide a fairly powerful (server-class) processor such as an Intel Xeon, 1-2 GB of memory, and perhaps a small SCSI disk. There can be several hundreds/thousands of these blade servers in a hosting center, all consuming electric power depending on whether they are turned on, and if so at what frequency. From the viewpoint of the customers (applications) of the hosting center, it is important to have at least as many servers as necessary for that application in order to meet a desired level of performance (SLA) for its client requests. In this exercise, we use a simple SLA that tries to bound the average response time for requests, and a more extensive SLA could have been used as well. From the viewpoint of the hosting center, the goal is to consume as little electrical energy as possible while still ensuring that the individual applications are able to meet their end-user SLAs for client requests. A schematic of the system environment is given in Figure 1.

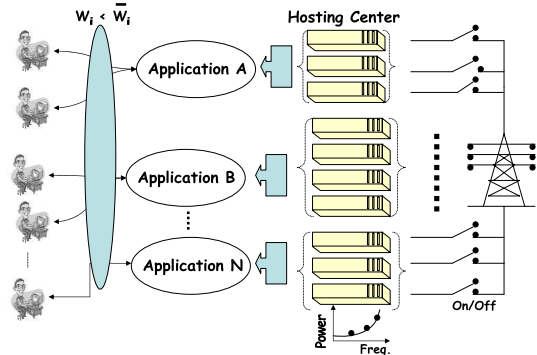


Figure 1: System Model

The workload imposed on a server's resources when running an application expends electric power, thereby incurring a cost in its operation. The important resources of concern include semiconductor components such as the server CPU, caches, DRAMs, and system interconnects, as well as electro-mechanical components such as disks. There are two mechanisms available today for managing the power consumption of these systems:

- One can temporarily power down the blade, which ensures that no electricity flows to any component of this server. While this can provide the most power savings, the downside is that this blade is not available to serve any requests. Bringing up

the machine to serve requests would incur additional costs, in terms of (i) time and energy expended to boot up the machine during which requests cannot be served, and (ii) increased wear-and-tear of components (the disks in particular) that can reduce the mean-time between failures (MTBF) leading to additional costs for replacements and personnel.

- Another common option for power management is dynamic voltage/frequency scaling (DVS). As will be shown in section 4, the dynamic power consumed in circuits is proportional to the cubic power of the operating clock frequency. Slowing down the clock allows the scaling down of the supply voltages for the circuits, resulting in power savings. Even though not all server components may be exporting software interfaces to perform DVS, CPUs [2] - even those in the server market - are starting to allow such dynamic control. With the CPUs consuming the bulk of the power in a blade server (note that an Intel Xeon consumes between 75-100 Watts at full speed, while the other blade components including the disk can add about 15-30 Watts), DVS control in our environment can provide substantial power savings.

Our framework allows the employment of both these options towards enhancing energy savings, without compromising on end-user SLAs. In our model, when a machine is switched off, it consumes no power. When the machine is on, it can operate at a number of discrete frequencies $f_1 < f_2 < \dots < f_\ell$, where the relationship between the power consumption and these operating frequencies is of the form $P = P_{fixed} + P_f \cdot f^3$ (see section 4), so that we capture the cubic relationship with the CPU frequency while still accounting for the power consumption of other components that do not scale with the frequency.

DVS implementation for a server cluster of these blades can be broadly classified based on whether (i) the control is completely decentralized, where each node independently makes its scaling choice purely on local information, or (ii) there is a coordinated (perhaps centralized) control mechanism that regulates the operation of each node. Though decentralized DVS control is attractive from the implementation viewpoint, previous research [15] has shown that a coordinated voltage scaling approach can provide substantially higher savings. In our system model, we use a coordinated DVS strategy where a controller is assigned for each application and it periodically assigns the operating frequency/voltage for the servers running that application. We perform this at an application granularity since the load for each application, and corresponding SLA requirements, can be quite different. Further, in our model, each server continues to be entirely devoted to a single application until the time of server reallocation.

With this model of the hosting center, the two solution steps to the problem at hand are to (i) perform server provisioning to decide how many servers to allocate to each application, and (ii) decide what should be the operating frequency for the servers allocated to each application. These steps need to be done periodically to account for time-varying behavior. Note that the first step may require bringing up/down servers and/or re-assigning servers. The time cost for such migrations of applications ($T_{migrate}$) and/or reboots (T_{reboot}) are incorporated in our model. Further, there is a long term impact of server reboots due to wear-and-tear of components such as the disk, and we include their dollar cost.

3. RELATED WORK

Resource Provisioning in Hosting/Data Centers:. Since many of the services/applications hosted at these centers can have

stringent service-level agreements to be met, there have been several investigations into resource capacity planning and dynamic provisioning issues for QoS control (e.g. [5, 31, 34, 38]). As many studies have pointed out, over-provisioning of resources can be economically unattractive, and it is important to accommodate transient overload situations (which are quite common for these services) with the existing resources [9]. Dynamic load monitoring (e.g. [30]), transient-load based optimization (e.g. [9, 36]) and feedback-based techniques (e.g. [3, 18, 26]) are being examined to handle these situations. However, these studies have mainly focused on performance (and revenue based on an SLA), and have not examined the power consumption issues.

Energy Management in Mobile Devices:. Energy management has traditionally been considered important for mobile and resource-constrained environments that are limited by battery capacities. One common technique is to shut off components (e.g. the disk [21],[24]) during periods of inactivity. The influence that the frequency (which allows the voltage to be scaled down) has on the power consumption has been exploited to implement Dynamic Voltage Scaling (DVS) mechanisms for energy management of integrated circuits [10], [32]. Consequently, many processors - not only those for the mobile market, but even those in the server space [2] - today are starting to offer software interfaces for DVS. There have been numerous studies on exploiting DVS for power savings without compromising on performance (e.g. [17], [19],[25], [28], [37]). However, all these studies have been for mobile devices that have very different workload patterns than servers, and/or for embedded environments where soft/hard real-time constraints need to be met within a power budget.

Energy Management in Servers and Data Centers:. It is only recently that energy management for server clusters found in data/hosting centers has gained much attention [8, 12, 16, 20, 23, 29]. Amongst these early studies, [11, 12, 29] show that resource allocation and energy management can be intertwined by developing techniques for shutting down servers that are not in use, or have low load (by offloading their duties to other servers). A detailed study of the power profile of a real system by researchers at IBM Austin [6] points out that the CPU is the largest consuming component for typical web server configuration. Subsequent studies [15, 33] have looked at optimizing this power, by monitoring evolving load and performance metrics to dynamically modulate CPU frequencies. A categorization of these most closely related investigations is summarized in Table 1, in terms of whether (i) the schemes consider just server shutdowns or allow DVS, (ii) they focus on energy management of just 1 server (which can be thought of as completely independent management of a server without regard to the overall workload across the system), (iii) they consider different applications to be concurrently executing across these servers with different loads, iv) they try to meet SLAs imposed by these applications, and (v) they incorporate the cost of rebooting servers (in terms of both time and MTBF).

Of the four related studies shown here, only two [15, 33] have considered the possibility of DVS for server applications. Of these two, only the IBM work [15] has examined the issues in the context of a server cluster, with the other focusing on a single machine/server setting. However, even the IBM work has not considered the issues in the context of multiple applications being hosted on these server clusters (i.e. the server provisioning problem in conjunction with energy management), nor have they considered the long term impact of machine reboots on operating costs.

	DVS?	Multiple servers?	Multiple applns.?	SLA?	Reboot cost?
Duke [11]	No	Yes(5)	Yes(2)	Yes	Yes(Time)
Rutgers [29]	No	Yes(8)	No	No	Yes(Time)
Virginia [33]	Yes	No	No	No	No
IBM [15]	Yes	Yes(10)	No	No	No
Ours	Yes	Yes	Yes	Yes	Yes

Table 1: Comparing the most related studies with this work. The numbers given in parenthesis indicate number of servers/applns. used in the study. The “Time” for the reboot cost indicates that only the time overhead of rebooting systems was considered (and not the wear-and-tear).

4. PROBLEM FORMULATION

We model our hosting centers to have M identical servers, all equally capable of running any application. Our model allows a maximum of N different applications to be hosted at any time across these servers, with each application i being allocated m_i servers at any time ($\sum_i m_i \leq M$). When a server is operational, it can run between a maximum frequency f_{max} (consuming the highest power) and a minimum frequency f_{min} , with a range of discrete operating frequency levels in-between. The service time of a request that comes to an application is impacted (linearly) by the frequency, and the overall service capacity allocated to an application is also directly related to the number of servers allotted to it.

A server CPU operating at a frequency f consumes dynamic power (P) that is proportional to $V^2 * f$, where V is the operating voltage [10]. Further, underlying circuit design inherently imposes a relationship between the operating voltage and circuit frequency. For instance, as we lower the voltage (to reduce power), the frequency needs to be reduced in order to allow for the circuits to stabilize. The common rule of thumb for this relationship is given by $V \propto f$. Since the power consumption of all other components (except the CPU) is independent of the frequency, we have the following simple model of power consumed by one cluster node running at frequency f : $P = P_{fixed} + P_f * f^3$. This cubic relationship between operating frequency and power consumption has been used in other related work as well [15]. We can then calculate the energy consumption of the servers at the hosting center operating at a constant frequency f , over time t , as

$$E = M * \int_t P * dt = M * (P_{fixed} + P_f * f^3) * t$$

Since the real workload is expected to be time varying, we need to control M and f over time to manage the energy consumption while adhering to any performance goals. Let us say that these are controlled periodically at a granularity of time t .

Electricity Cost of Operation.: Over a duration of Z such time units of duration t , the overall energy consumption is $\sum_{z=1}^Z \sum_{i=1}^N m_i(z) * (P_{fixed} + P_f * f_i(z)^3) * t$, where $m_i(z)$ is the number of servers allocated to application i during the z -th time period, which are all running at a frequency $f_i(z)$. If $K_{\$}$ is the electricity charge expressed as dollars per unit of energy consumption (e.g. kilowatt hour), then we have the total electricity cost (in dollars) for operation of the hosting center to be $\sum_{z=1}^Z \sum_{i=1}^N K_{\$} * m_i(z) * (P_{fixed} + P_f * f_i(z)^3) * t$

Cost of Server Turn-ons.: One important point to consider is the impact of turning on/off servers. While frequency scaling is itself relatively simple (taking only a few cycles), and may not

significantly affect the long term reliability of the system, the effects of turning off/on the servers should not be overlooked. Machine reboots can last several seconds/minutes, consuming power in the process. Further, it is well understood that server components (disks in particular) can be susceptible to machine start-stop cycles [14], thus reducing the MTBF when we perform server shut-downs. If B_o is the dollar cost of a single server turn-on cycle, then the total amount of dollars expended by any mechanism using this approach can be calculated as

$$\sum_{z=1}^Z B_o * \left[\sum_{i=1}^N m_i(z) - \sum_{i=1}^N m_i(z-1) \right]^+$$

Note that the term $x^+ = x$ when $x \geq 0$ and $x^+ = 0$ otherwise. B_o itself needs to account for the dollar energy cost for bringing up the machine ($P * T_{reboot} * K_{\$}$), together with the dollar cost incurred by a smaller MTBF (denoted as C_r), i.e. $B_o = P_{max} * T_{reboot} * K_{\$} + C_r$, where P_{max} denotes the server power consumption when running at f_{max} .

Putting these together, we get the dollar cost of operation to be

$$\sum_{z=1}^Z \left(\sum_{i=1}^N K_{\$} * m_i(z) * (P_{fixed} + P_f * f_i(z)^3) * t + B_o * \left[\sum_{i=1}^N m_i(z) - \sum_{i=1}^N m_i(z-1) \right]^+ \right)$$

which is the *objective function to minimize*. Note that one could choose to minimize this objective by inordinately slowing down the frequencies, or shutting down a large number of servers. This can result in violating any service level agreements (to meet a response time requirement) to customers/applications. We use a simple SLA, where the requirement is to meet an average response time target. Consequently, when minimizing the above objective function, we need to obey the following constraints:

- $W_i \leq \bar{W}_i$, which is the SLA to constrain the average response time of application i to a target \bar{W}_i .
- $f_i(z) \in \mathcal{F} = (f_1, f_2, \dots, f_\ell)$ where $f_1 < f_2 < \dots < f_\ell$, which says that a running server has to operate at one of the ℓ discrete frequencies. We use f_{min} to denote f_1 and f_{max} to denote f_ℓ .
- $\sum_i^N m_i(z) \leq M$, since the total number of allocated servers has to be less than total capacity.

5. METHODOLOGY

We next discuss three mechanisms to determine the number of servers (m_i) allocated to each application i and their frequency (f_i) at any instant. Note that all m_i servers of application i run at the same frequency f_i as explained in section 2. To accommodate time-varying workload behavior, we assume that every T minutes, servers are allocated for the N applications (i.e. for all i , we determine m_i). Likewise, every t minutes, we determine the frequency f_i for the m_i servers of application i . Although not required for modeling, for logical and implementation reasons we assume that T is an integer multiple of t . Thereby at larger time granularities T , we perform server allocation/provisioning, and at smaller time granularities (t), we tune the servers to different frequencies. The basic premise for doing this at 2 granularities, with $T > t$ is that server re-allocation (machine reboots and application migration) is much more time-consuming than changing the frequency. There are U intervals of length T present in the entire duration for which we want to optimize, and each of these (denoted by $u = 1$ to U) has S intervals of length t (i.e. $s = 1$ to S). This is pictorially shown

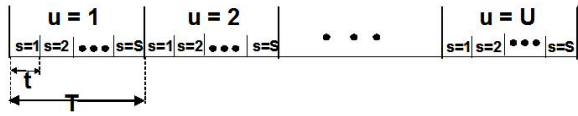


Figure 2: Granularity (T, t) for server allocation and frequency modulation

in figure 2. We first propose two techniques - non-linear optimization based on steady-state queuing analysis, and a technique based on control theory - to determine f_i and m_i at every t and T minutes respectively. Then, we illustrate a hybrid approach integrating these techniques.

5.1 Queuing Theory Based Approach

We can model the system under consideration as a set of N parallel queues denoting the N applications. Client requests for application i arrive one by one into its corresponding infinite capacity queue (queue- i). Setting m_i and f_i by analyzing this queuing system involves three phases. First, we need to predict request arrival pattern and service time requirements. Next, we can use the predicted arrival and service time information in a queuing model to determine the mean response time for each application. Finally, we can use the response time information in our optimization problem, for which we describe a tractable solution strategy.

5.1.1 Prediction

To perform queuing analysis in each interval, we require estimates of mean arrival rate of requests (λ), squared coefficient of variation of request inter-arrival times (C_a^2), mean file size in bytes (ϕ), and squared coefficient of variation of file size (C_s^2) for each application i . Notice that all four of the above parameters are time varying as well as stochastic, requiring a prediction of their values at each interval. Historically, researchers have used either self-similar traffic patterns or used deterministic time-varying parameters. On one hand, self-similar models are attractive for describing the nature of the traffic and generating synthetic workloads, but they do not lend themselves well for analysis. On the other hand, time-varying models require Poisson assumptions and full knowledge of the time-varying behavior. *It is important to note that it is not our goal here to determine the best prediction technique.* However, our contribution is that if the parameters are well predicted, we can approximate the inter-arrival times and file size requirements as independent and identically distributed (i.i.d.) inside each interval and use this conveniently for online optimization.

We present a simple prediction technique with the disclaimer that this can surely be improved upon. From one time interval to another, for the mean inter-arrival times and standard deviation of inter-arrival times, we use a multiplicative S-ARMA (seasonal autoregressive moving average) [35]. We use several intervals of “training” data to tune the S-ARMA model. Then, based on the cumulative average, seasonality effects, as well as the value at a few previous intervals, we identify the parameters for a given interval. For mean and standard deviation of file size we use a simple decomposed model or Winter’s smoothing method [35]. In general, we found that the prediction works reasonably well for arrival rates, but the errors can be higher in terms of file sizes in certain cases. However, as we mentioned, our goal here is not necessarily to derive the best prediction mechanism. Note that the prediction values will be computed dynamically at the beginning of each interval during the actual execution with the real data.

5.1.2 Queuing Analysis

Using the predictions of $\lambda(i)$, $C_a^2(i)$, $\phi(i)$, and $C_s^2(i)$ for a given interval for each application i , we next obtain an expression for the predicted average response times (W_i) in that given interval. We approximate queue- i to be a $G/G/m_i$ queue with i.i.d. inter-arrival times and i.i.d. service times. Further, we assume that the time interval is large enough that steady state results can be used (note that this is one of the reasons why results with this technique may not be very good for small time granularities of control as our evaluations will show). There are several approximations for response time in the literature, of which many are empirical. In this study, we use the method in Bolch et al [7], which states that

$$W_i = \frac{\phi(i)}{\beta * f_i} + \frac{\alpha_{m_i} \phi(i)}{\beta * f_i} \left(\frac{1}{1 - \rho_i} \right) \left(\frac{C_a^2(i) + C_s^2(i)}{2m_i} \right), \quad (1)$$

where $\beta * f_i$ is the bandwidth of the server in bytes served per second for application i (β is a constant that is calculated by measuring the service time of HTTP requests of different sizes on an actual system), $\rho_i = \frac{\lambda(i)\phi(i)}{m_i * \beta * f_i}$, and $\alpha_{m_i} = \rho_i^{\frac{m_i+1}{2}}$. Notice that W_i in Equation (1) is non-linear with respect to the decision/control variables f_i and m_i . Also, we do not need to write down the constraint $\rho_i < 1$ explicitly in the optimization problem as satisfying the SLA constraint would automatically ensure this.

Note that W_i decreases with respect to both m_i and f_i . Therefore the SLA constraint would be binding if f_i and m_i were continuous. However in the discrete case, we can obtain an efficient frontier (i.e. for every f_i we can find the smallest m_i that will render the SLA constraint feasible). As there are ℓ levels for f_i , we will have to consider only ℓ pairs of m_i and f_i for each application in each interval, as is exploited below.

5.1.3 Solving the optimization problem

We use s and u in all the parameters to denote the corresponding subscripts for S and U intervals as shown in Figure 2. For example, $m_i(u)$ is the number of servers in queue- i at the u^{th} interval of duration T , and $f_i(u, s)$ is the frequency of a server in queue- i at the s^{th} interval of duration t for this given u . The optimization problem can then be re-written in terms of the decision variables $f_i(u, s)$ and $m_i(u)$ (assume for all i , $m_i(0) = 0$) as:

$$\begin{aligned} \min_{f_i(u,s), m_i(u)} & \sum_{u=1}^U \left(\sum_{s=1}^S \sum_{i=1}^N K_{\$} m_i(u) (P_{fixed} + P_f * f_i(u, s)^3) t \right. \\ & \left. + B_o \left[\sum_{i=1}^N m_i(u) - \sum_{i=1}^N m_i(u-1) \right]^+ \right) \\ \text{subject to} & \\ & W_i \leq \bar{W}_i \text{ for all } i \\ & \sum_{i=1}^N m_i(u) \leq M \text{ for all } u, \text{ and } m_i(u) \text{ is a non-negative integer} \\ & f_i \in \mathcal{F} \end{aligned}$$

Clearly, the above optimization is non-linear and discrete in terms of the decision variables for both the objective and the constraints. Hence standard optimization techniques are not enough. Since we only consider a finite number of frequency alternatives, it is tempting to do a complete enumeration to determine the optimal solution. However, a complete enumeration would require comparing $O(\ell^S M^U N)$ values - and performing these periodically during execution. Therefore we resort to a heuristic as explained below.

We revise the prediction model described above dynamically based on the previous observation (for $u - 1$), and then run the following two steps at the beginning of each u . We explain the steps briefly without going into details. We first consider the case $t = T$, and therefore for all intervals $s = 1$. Under this framework, we use $f_i(u, s) = f_i(u)$ for ease of notation.

Step 1: We first obtain a feasible solution that would result in an upper bound to the objective function. We consider only a single interval and optimize the parameters $f_i(u)$ and $m_i(u)$ for the simplified objective as below:

$$\min_{f_i(u), m_i(u)} \sum_{i=1}^N K_g m_i(u) (P_{fixed} + P_f * f_i(u)^3) t + B_0 \sum_{i=1}^N m_i(u)$$

There are two ways of solving this, one is to assume the decision variables ($m_i(u)$ and $f_i(u)$) are continuous and use standard non-linear programming techniques. This is especially useful when the problem is scaled to a large number of applications. However, we use a second method where we exploit the monotonicity properties of the objective function and constraints as well as the fact that the frequencies take only a small number of discrete values. For that purpose, we use estimates of moments of inter-arrival times and file sizes, across all applications and across all intervals. Then, for each interval u we start by finding the minimum number of servers $m_i(u)$ for all applications i so that the constraint $W_i(u) \leq \bar{W}_i$ can be satisfied using the highest frequency for the servers. Note that the total number of servers in the hosting center M is large enough that this would guarantee the constraint $\sum_{i=1}^N m_i(u) \leq M$ to be automatically satisfied - otherwise it implies the SLA has been chosen poorly since it would have been violated even without any energy management. This solution can be improved by recognizing that the objective is a cubic in terms of the frequency that needs to be reduced. Therefore by suitably decreasing $f_i(u)$ and increasing $m_i(u)$, as long as the constraints are satisfied, we can obtain a solution that would be close to the optimal solution, i.e. we are trying to find the number of servers that are needed when they are all operating at lower frequencies that can give the most power savings. When reducing $f_i(u)$, we select applications in decreasing order of $f_i(u)$ for each interval.

Step 2: We next consider all the intervals together. The actual objective function that is the total cost across the entire execution, is used for this purpose. We use the upper bound solution from the previous step and work our way accepting feasible solutions that reduce the objective function value. If a point in this feasible space improves the objective, we accept that solution and search further in its neighborhood. In our approach we consider one interval at a time (going from the first to the last) and in each interval we select the applications in the decreasing order of their frequencies. For each application we compare the number of servers in that interval against those of the next interval. We try to level off the number of servers (to the extent possible) whenever the resulting solution improves. We search greedily giving importance to the points where the number of servers for this interval are close to the number for the previous interval (to reduce turn-on costs). Thereafter, we tune the frequencies so that the resulting solution remains feasible.

Note that $T = t$ is being assumed in the above two steps. To handle the cases where $T > t$, we first perform the above optimization (assuming $t = T$ and using average values of arrival and file size parameters over every interval) at each of $u = 1$ to U , to first determine the $m_i(u)$. Subsequently, using the prediction information for each s , we determine the appropriate $f_i(u, s)$ using this pre-determined value of $m_i(u)$.

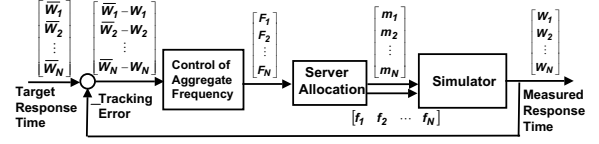


Figure 3: Overview of the control-theoretic approach

5.2 Control Theoretic Approach

An alternative strategy that we propose is based on feedback control theory. Compared to the previous mechanism that relies on steady-state analysis of the system, control theory provides a way of addressing the system's transient dynamics using feedback. It can allow finer granularity of controlling the system, to become more responsive to workload changes. In this approach, we decompose the given problem into two sub-problems:

1. Dynamically determine an aggregate frequency for each application that can meet response time guarantees when there is a single server per application running at this frequency; the objective for this subproblem is to meet response time with minimal aggregate capacity.
2. Solve a server allocation problem, which determines the number of servers for each application in order to provide the aggregate frequency obtained from the first subproblem. The objective for this subproblem is to balance between the cost of turning on new servers and the energy consumption of running servers at a higher frequency.

Figure 3 shows a schematic of this approach. At each time period t , based on the feedback of tracking error that is the difference between the measured response time and its target value, an aggregate frequency will be computed for each application using control theory to provide response time guarantees. The number of servers for each application will then be allocated through an on-line optimization to provide the required aggregate frequency. The frequency for each individual server is calculated in terms of the aggregate frequency and the number of running servers. Since we set all servers for each application to be running at the same frequency f_i , the aggregate frequency provided for the i -th application is calculated as $F_i(u, s) = m_i(u) * f_i(u, s)$ for $u = 1, \dots, U; s = 1, \dots, S$. We use $F_i(u, s)$ interchangeably with $F_i(k)$, $k = 1, \dots, U * S$ when necessary. It should be noted that though the notion of aggregate frequency is based on approximating the response time from multiple servers by the response time from a single server with the same capacity, the feedback of real response time will hopefully make enough adjustments for this approximation. The details of the feedback control block and the on-line optimization block are given next.

5.2.1 Feedback Control of Aggregate Frequency

For the first subproblem, we apply optimal control theory to dynamically determine the aggregate server frequency for each application. The original formulation in Section 4 minimizes operation cost subject to a constraint on response time. In this section, we incorporate the response time SLA into the objective function to meet response time with as low cost as possible. We modify the objective function as follows,

$$\sum_{i=1}^N \sum_{k=1}^{U*S} (R_F * F_i^3(k) + R_{W,i} * (W_i(k) - \bar{W}_i)^2) \quad (2)$$

where k is the index for time interval t ; R_F and $R_{W,i}$ are weights whose ratio provides tradeoffs between meeting target response

time and having lower energy cost. The weight parameters R_F and $R_{W,i}$ are chosen to give priority to the SLA. The aggregate frequency F_i is constrained by the total system capacity, i.e.

$$\sum_{i=1}^N F_i(k) \leq M * f_{max} \quad (3)$$

for $k = 1, \dots, U * S$. The cost function (Eq. 2) together with constraint (Eq. 3) define a constrained multi-input (F_i) multi-output (W_i) optimal control problem. One way to deal with the constraint (Eq. 3) is to incorporate it into the cost function. In this paper, since we assume that there is enough computation capacity to meet the response time guarantee for all applications (otherwise, the SLA would not have been agreed upon), constraint (Eq. 3) is always satisfied. Consequently, we solve the aggregate server frequency for each application separately, which leads to solving N independent single-input single-output control design problem.

From the optimal control literature, the Linear Quadratic (LQ) regulator [4] seems to be the most appropriate way of solving this problem. However, in a general LQ formulation, the cost function depends on a quadratic term of the control variable. Therefore, we define a new control variable $\tilde{F}_i = F_i^{3/2}$ to make the LQ method applicable for our problem. Based on the above arguments and approximations, the cost function for optimal control design is defined as follows

$$J_i = \sum_{k=1}^{U*S} (R_F * \tilde{F}_i^2(k) + R_{W,i} * (W_i(k) - \bar{W}_i)^2) \quad (4)$$

for $i = 1, \dots, N$. Next, we first use system identification techniques to build a dynamic model between the new control input \tilde{F}_i and the response time W_i , after which a LQ control law is derived.

System Identification. We approximate the dynamic relation from \tilde{F}_i to the response time W_i by a linear second-order ARX model as follows:

$$W_i(k+2) = a_{i,1}W_i(k+1) + a_{i,2}W_i(k) + b_{i,1}\tilde{F}_i(k+1) + e_i(k+2) \quad (5)$$

or in a state-space form,

$$\begin{pmatrix} W_i(k+1) \\ W_i(k+2) \end{pmatrix} = H_i \begin{pmatrix} W_i(k) \\ W_i(k+1) \end{pmatrix} + G_i \tilde{F}_i + \begin{pmatrix} 0 \\ e_i(k+2) \end{pmatrix} \quad (6)$$

where $H_i = \begin{pmatrix} 0 & 1 \\ a_{i,2} & a_{i,1} \end{pmatrix}$, $G_i = \begin{pmatrix} 0 \\ b_{i,1} \end{pmatrix}$, and e_i denotes the noise in the system, $k = 1, \dots, U * S - 2$.

A second-order model is chosen here as a balance between model complexity versus accuracy to fit the empirical data. For system identification, a pseudo-random binary signal is used to generate the server frequency trajectory \tilde{F}_i , and part of the input workload is used to produce the corresponding response time trajectory W_i ; then the data set (\tilde{F}_i, W_i) is used by standard system identification toolbox in Matlab to compute the model coefficients in (Eq. 5). It should be noted that even though we use a real trace of HTTP requests for system identification (which is the same as that used in generating prediction models for queuing in section 5.1.1), this trace is for a time period that is different from what is used in the actual simulations and evaluations. Since the coefficient values in Equation 5 depend on the sampling granularity of the empirical data, for each time granularity (T,t) that we use in our evaluations, an individual system model (Eq. 5) is constructed for control design. Details on the accuracy of the model can be found in [13].

Control Law. Given the cost function (Eq. 4) together with the linear dynamic equation (Eq. 6), the Linear-Quadratic regulator provides an optimal solution that is always stabilizing. The resulting LQ control input \tilde{F}_i is represented as the product of an optimal feedback gain with the tracking error in meeting response time,

$$\tilde{F}_i(k+1) = -R_F^{-1} \Pi_i G_i \begin{pmatrix} W_i(k) - \bar{W}_i \\ W_i(k-1) - \bar{W}_i \end{pmatrix} \quad (7)$$

In terms of the LQ control theory, the negative feedback gain $-R_F^{-1} \Pi_i G_i$ is determined by solving Π_i from the following Riccati equation,

$$\begin{aligned} H_i^T \Pi_i H_i - \Pi_i - (H_i^T \Pi_i G_i)(R_F + G_i^T \Pi_i G_i)^{-1} (G_i^T \Pi_i H_i) \\ + R_{W,i} = 0 \end{aligned} \quad (8)$$

The LQ solution can be easily calculated using existing Matlab toolbox. The aggregate frequency F_i for the i -th application is then calculated by $F_i = \tilde{F}_i^{2/3}$. In the implementation of the LQ regulator, an integrator is appended to reduce the steady-state tracking error in meeting response time SLA. Thus overall controller is

$$\begin{aligned} F_i(k+1) &= F_i(k) + \tilde{F}_i^{2/3}(k+1) \\ &= F_i(k) - R_F^{-2/3} \left\{ \Pi_i G_i \begin{pmatrix} W_i(k) - \bar{W}_i \\ W_i(k-1) - \bar{W}_i \end{pmatrix} \right\}^{2/3} \end{aligned} \quad (9)$$

for $k = 1, \dots, U * S - 1$. The weights R_F and $R_{W,i}$ in (Eq. 4) are design parameters. The rule of thumb is to set R_F as $1/(\tilde{F}_{i,max})^2$ and $R_{W,i}$ as $1/(W_{i,max} - \bar{W}_i)^2$, where the subscript *max* denotes the maximum possible value.

5.2.2 Server Allocation

Given the aggregate frequency $F_i(u, s)$, an on-line optimization is formulated to allocate servers to each application. This is based on the tradeoff between the energy cost of running servers with higher frequencies versus the cost of turning on more servers.

Define $m(u) = \sum_{i=1}^N m_i(u)$, which denotes the total number of servers being turned on across all applications, and define $F(u) = \sum_{i=1}^N (\max_{s=1, \dots, S} F_i(u, s))$, which denotes the total capacity that $m(u)$ should provide at each time interval of T . We allocate the number of servers for each application $m_i(u)$ proportional to its aggregate frequency, i.e.,

$$m_i(u) = \lceil m(u) * \frac{\max_s F_i(u, s)}{F(u)} \rceil \quad (10)$$

Since server allocation occurs at $u = 1, \dots, U$, by using $f_i(u, s) = F_i(u, s)/m_i(u) \leq F(u)/m(u)$ in the cost function defined in Section 4, we minimize the following cost,

$$\begin{aligned} \sum_{u=1}^U (K_{\$} * m(u) * T * (P_{fixed} + P_f * (F(u)/m(u))^3)) \\ + \sum_{u=1}^U B_0 * [m(u) - m(u-1)]^+ \end{aligned} \quad (11)$$

Each server's frequency should be operated within $[f_{min}, f_{max}]$. Consequently, for providing the aggregate frequency $F(u)$, the total number of servers that are allowed to run across all applications at u is constrained by $\underline{m}(u) \leq m(u) \leq \bar{m}(u)$, where $\underline{m}(u) = \lceil \frac{F(u)}{f_{max}} \rceil$ and $\bar{m}(u) = \min(\lfloor \frac{F(u)}{f_{min}} \rfloor, M)$.

Equation (11) is a cost function with decision variable $m(u)$. We compute $m(u)$ at the beginning of each u , but at that time the value of F calculated based on the feedback control in section 5.2.1 is

only up to $F(u-1)$ and the values of $F(u)$ to $F(U)$ are not available. Therefore, an on-line greedy algorithm is designed to solve $m(u)$ in minimizing Equation (11). Further, we use the capacity from the previous interval $F_i(u-1, s)$ and $F(u-1)$ to calculate $m_i(u)$ in Equation (10). A pseudo-code of this on-line algorithm can be found in [13].

The general idea behind this on-line server allocation algorithm is explained as follows. If we ignore the server turn-on costs in Equation (11), minimizing the number of servers (denoted by $m^*(u)$) for the energy cost can be achieved by setting the first derivative of Equation (11) with respect to $m(u)$ as zero, i.e.

$$\begin{aligned} \frac{\delta}{\delta m(u)} (K_{\S} * m(u) * T * (P_{fixed} + P_f * (F(u)/m(u))^3) \\ = K_{\S} * T * (P_{fixed} - \frac{2P_f * F^3(u)}{m^3(u)}) = 0 \end{aligned} \quad (12)$$

which gives

$$m^*(u) = \lfloor F(u) * (\frac{2P_f}{P_{fixed}})^{1/3} \rfloor \quad (13)$$

This $m^*(u)$ gives the break even number of servers based on the trade-off between the increase in cost due to turning on more servers, and the corresponding decrease in power expended because of allowing them to operate at a lower frequency. If we add one more server beyond $m^*(u)$, the increase in cost due to the addition will outweigh the cost reduction due to the decrease of server frequency. Therefore, the number of servers being turned on $m(u)$ should be always less than $m^*(u)$ even without considering the cost of server turn-ons.

If we take into account the cost of turning on servers (B_o), at any $u, u = 1 \dots, U$, there are two cases:

1. If the number of running servers in the previous time period $m(u-1)$ is higher than the minimum of the break-even number $m^*(u)$ and the upper-bound $\bar{m}(u)$, the number of servers $m(u)$ should be brought down to $\min(m^*(u), \bar{m}(u))$ - there is no associated boot cost for this.
2. Otherwise, additional servers beyond $m(u-1)$ could be turned on, but the total number of running servers is still upper-bounded by $\min(m^*(u), \bar{m}(u))$.

For the second case, the number of servers is determined iteratively. Assuming that the number of servers has been increased from $m(u-1)$ to i , turning on one more server beyond i will add a one-time boot cost B_o , while it will reduce power cost by $K_{\S} * (j-u+1) * T * (-P_{fixed} + \frac{2P_f F^3(u)}{i^3})$ (which is the derivative of the first term in Equation (11) at $m(u) = i$) only if no server will be turned off during time u to time j . The latter condition is checked by verifying if i is less than the minimum of $m^*(u)$ and $\bar{m}(u)$ up to time j . Note that at current time period u , we do not have future aggregate frequency $F(j)$ for $j \geq u$ to calculate $\bar{m}(j)$ - we use an estimated value $\bar{m}(j)$ to restrict i - which could cause the algorithm to become more greedy. Based on this argument, the algorithm searches if there exists such j within the whole U time periods during which the saving in power cost is larger than the one-time boot cost for turning on one more server. More servers will be turned on when such j exists until the number of running servers touches its upper bound.

After the aggregate frequency and number of servers for each application (Equation (10)) has been determined, the server frequency is calculated as

$$f_i(u, s) = \mathcal{D}(\frac{F_i(u, s)}{m_i(u)}) \quad (14)$$

for $u = 1, \dots, U$ and $s = 1, \dots, S$, where $\mathcal{D}(\cdot)$ denotes rounding up the frequency to discrete levels in $\mathcal{F} = (f_1, f_2, \dots, f_\ell)$.

5.3 Hybrid Approach

The queuing approach, described above, can be viewed to be “pro-active”, since it predicts workload behavior for the near future and tunes server allocation and frequency setting appropriately. While it can do well when the predictions are fairly accurate, and the steady state stochastic behavior is obeyed (requiring a fairly coarse-grained window), it may not be adaptive enough for fine-grained transient behavior. On the other hand, the control theoretic approach can be viewed to be “reactive” since it bases its decisions on feedback (we are using feedback control here rather than feed-forward), and is thus expected to be better at finer granularities, though it can possibly miss out on the predictive information for a longer granularity of optimization. This leads us to consider a hybrid scheme, where we use the predictive information of the queuing approach to determine server allocation (section 5.1) at a granularity of T , and the feedback-based control theoretic approach for frequency setting (section 5.2.1) at the smaller granularity of t . Note that this strategy is also in agreement with our underlying philosophy where server provisioning costs (in terms of time for server turn-on and application migration) are anticipated to be much higher than frequency control, and are thus expected to be done less frequently (i.e. $T > t$).

6. EXPERIMENTAL EVALUATION

6.1 Experimental set up

We used real HTTP traces obtained from [1] during the last week of September and early October 2004, for our evaluation, whose arrival time characteristics are shown in Figure 4. There are 3 traces in all, denoted as Applications 1 to 3, with each trace being for a 3 day duration. We use the data of the first 2 days (called training data) to build the prediction model for the Queuing approach and the system identification model for the control-theoretic approach, while only the 3rd day’s trace is used in the actual simulation/evaluation. We mix these three applications in our experiments to get different workloads, WL1 to WL3: WL1 contains only one application, namely Application 2; WL2 includes both Applications 1 and 2; while WL3 includes all 3 applications.

The techniques have been evaluated using a simulator built on top of the CSIM simulation package. The simulator takes the server frequency f_i and the number of servers m_i for each time period as inputs, and generates response time W_i as system output for a given static HTTP request trace. During the simulation, it also calculates the energy consumed, number of reboots, and cost of operation. The cost of reboots and migrating a server from one application to another are inputs to the simulator. In the interest of space, we give a brief description of the simulation model below, and the reader is referred to [13] for further details.

Since we are primarily interested in the CPU power, which as explained in section 2 dominates over the other components, we assume that the static HTTP requests hit in the cache. Further, as pointed out in [15], 99% of the HTTP requests can be easily served from the cache. We ran microbenchmarks of requests with different file sizes on a server machine that hit in the cache and used this as the service times for the requests at the highest operating frequency (2.6 GHz). As expected, the relationship between file sizes and service times was more or less linear. We also conducted similar experiments on a laptop with DVS capabilities to confirm that the service time is inversely proportional to the operating frequency of the CPU, and appropriately adjusted the service times in our sim-

ulation model for the server class CPU. Each node in the simulated cluster uses these service times in serving requests in FCFS order. The maximum power consumption at the highest frequency in our simulator is 100 W, which roughly matches the consumption on current server CPUs. We then use the well-known cubic relationship between frequency and power, to model the power at each discrete frequency, which is similar to the technique in [15].

We restrict the results presented here to the parameter values shown in Table 2 in the interest of space. The table gives the different possible CPU frequencies, and their associated power consumption values. The electricity cost (K_S) is close to that being charged currently, and the dollar cost associated with wear-and-tear per reboot is based approximately on the cost of a disk replacement (say \$200 including personnel) and the rated MTBF (say 40000 start-stop cycles). Though our model allows per application \bar{W}_i , in the interest of clarity we use the same target response time for all applications of 6 ms. We have ensured that this is achievable for each workload by provisioning an appropriate number of servers (M) as given in the table. We consider different combinations of T and t , with $t < T$, i.e. DVS being done much more frequently than server allocation, with (T, t) being used to denote the experiment.

Parameter	Value
\mathcal{F} (in GHz)	(1.4, 1.57, 1.74, 1.91, 2.08, 2.25, 2.42, 2.6)
P (in Watts for each f)	(60, 63, 66.8, 71.3, 76.8, 83.2, 90.7, 100)
K_S (cents/KWH)	10
C_r (cents/boot)	20,000/40,000 = 0.5
B_o (cents/boot)	$C_r + 0.025 = 0.525$
$T_{reboot}, T_{migrate}$	90 secs, 20 secs
M	13 (for WL1), 31 (for WL2), 39 (for WL3)
\bar{W}_i (in msec)	6
T (in minutes)	(60, 20, 10)
t (in minutes)	(60, 20, 10, 5) with $t \leq T$

Table 2: Default Parameter Values for Simulation

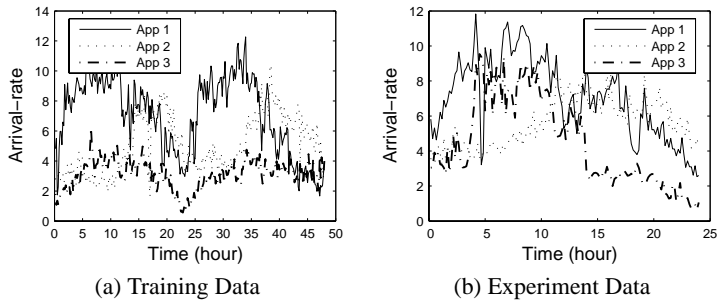


Figure 4: Time Varying Arrival Rates

Metrics: The main statistics that we present here include (i) the objective function (i.e. *Cost*), (ii) the energy consumption (*Energy %*) expressed as a percentage of the energy that would be consumed if all the servers were running all the time at the highest frequency, i.e. without any power management, (iii) the number of reboots (*#reboots*), and (iv) the mean response time (*RT*). When evaluating any scheme we want to first check whether it is able to meet the response time SLA (i.e. a target of 6 ms), before checking its energy consumption or cost. However, it should be noted that some schemes may very marginally violate this SLA while saving substantial cost. In order to not penalize such a scheme, we allow a 10% slack in the SLA, i.e. we consider schemes that can go as high as 6.6 ms in its mean response time when comparing them, and we term such a scheme to be a *viable* option.

6.2 Results

6.2.1 Need for Sophisticated Strategies

Before evaluating our schemes, we first present some results to motivate their need, by examining the following three “simple” schemes running WL1:

1. *Fixed servers, constant frequency during the entire experiment:* Results with such a scheme for the maximum servers (=13) by running the experiment in a brute-force fashion with each frequency level is shown in Table 3 (a). If we want any “viable” option with this scheme (i.e. $RT < 6.6$), the lowest cost we can get is 196 cents, which is around 15% higher than what Queuing can provide at (60,60) - see Table 4 - and we can do even better at other granularities.
2. *Highest frequency, constant number of servers during the entire experiment:* This corresponds to statically configuring the hosting center with the minimum number of servers needed to meet the SLA if they were all to run at the maximum speed. Looking at the results for this scheme in Table 3 (b), we see that it takes at least 8 servers operating at full frequency, before this becomes a “viable” option. Even at this number of servers, the energy consumption and cost are higher than for our schemes at different time granularities (compare with Table 4).
3. *Maximum number of servers, switch between f_{min} and f_{max} :* Whenever there is no request, a server switches to the minimum frequency (no cost is assumed for such switching), and whenever a request arrives it switches to the maximum frequency until it has finished serving the request. Clearly, the response time is not any different from operating at maximum frequency. However, the energy (72.9% of having them all operate at highest frequency) and cost (227.55 cents) are much higher than what we can get with our schemes.

Note that the first two schemes require a brute-force evaluation of all alternatives, and are thus not suitable in practice. Our point in showing their results was to point out that even if one had a very good static server capacity allocation mechanism for the hosting center, results with a dynamic power management strategy can provide better savings due to time-varying behavior of the workload. One could envision the above third scheme to be a “no-brainer” power management strategy if there are no costs to transition between frequencies. However, with reasonable load conditions, there is not that much scope for operating at lower frequencies. The reader should note that we could use any of our three proposals in conjunction with this third simple mechanism in order to set the maximum frequency at which to serve a request dynamically (and to otherwise bring it down to f_{min}). However, we do not consider that in the evaluations.

Freq. (GHz)	Cost (cents)	RT (ms)	Energy (%)	Serv. (#)	Cost (cents)	RT (ms)	Energy (%)
1.40	187.20	9.08	60.00	1	24.00	29368.20	7.69
1.57	196.76	5.07	63.07	2	48.00	17846.13	15.38
1.74	208.66	3.25	66.88	4	96.00	3065.25	30.77
1.91	223.14	2.34	71.52	6	144.00	28.45	46.15
2.08	240.46	1.87	77.07	8	192.00	5.05	61.54
2.25	260.87	1.57	83.61	10	240.00	1.95	76.92
2.42	284.63	1.36	91.23	12	288.00	1.35	92.31
2.60	312.00	1.20	100.00	13	312.00	1.20	100.00

(a) Fixed Servers = 13, Const. Freq.

(b) Max. Freq. = 2.6 GHz, Const. Servers

Table 3: Results for the “simple” strategies

(T,t)	Schemes	WL1				WL2				WL3			
		Cost (cents)	RT (ms)	Energy (%)	#reboots	Cost (cents)	RT (ms)	Energy (%)	#reboots	Cost (cents)	RT (ms)	Energy (%)	#reboots
(60, 60)	Queueing	✓ 170.00	5.86	51.54	16	✓ 337.96	4.91	42.90	34	✓ 492.90	5.81	50.26	44
	Control	✓ 196.75	4.89	60.00	9	✓ 351.97	5.70	44.84	15	✓ 523.42	5.77	53.33	24
	Hybrid	✓ 191.68	5.47	58.46	16	✓ 346.70	5.86	44.19	35	✓ 508.97	5.83	51.79	44
	IBM	110.55	11.07	35.38	18	210.57	8.25	28.39	34	310.06	11.80	33.08	49
(60, 20)	Queueing	✓ 161.55	6.52	49.23	16	✓ 345.14	5.97	43.87	34	✓ 487.47	5.68	49.49	44
	Control	186.84	6.77	56.92	12	346.41	7.08	44.19	16	✓ 512.59	6.19	52.31	24
	Hybrid	✓ 185.52	5.79	56.92	16	✓ 343.11	6.55	43.55	35	✓ 502.96	5.82	51.28	44
(60, 10)	Queueing	✓ 162.62	5.96	49.23	16	✓ 345.48	6.31	44.19	34	✓ 489.70	5.41	49.74	44
	Control	187.00	6.62	56.92	15	343.55	7.16	43.55	17	✓ 506.65	5.81	51.54	23
	Hybrid	✓ 184.06	5.82	56.15	16	✓ 339.64	5.94	43.23	35	✓ 498.23	5.45	50.77	44
(60, 5)	Queueing	158.69	12.84	48.46	16	324.61	10.93	41.29	34	469.17	8.47	47.69	44
	Control	✓ 187.40	6.58	57.69	13	342.30	6.94	43.55	18	✓ 509.12	5.78	52.05	24
	Hybrid	✓ 183.52	6.37	56.15	16	✓ 344.81	6.33	43.87	35	✓ 506.19	5.85	51.54	44
(20, 20)	Queueing	✓ 167.41	6.45	48.46	32	✓ 363.47	5.85	44.19	67	✓ 495.98	5.58	48.97	72
	Control	✓ 195.16	5.98	56.92	30	✓ 365.74	6.19	43.87	19	✓ 522.73	5.27	51.79	29
	Hybrid	✓ 190.06	4.92	55.38	32	✓ 361.90	5.33	43.23	77	✓ 496.91	5.20	48.97	72
	IBM	95.51	38.85	30.77	25	178.57	34.73	24.19	49	261.78	38.06	28.21	69
(20, 10)	Queueing	✓ 169.72	6.10	49.23	32	✓ 367.39	4.84	44.52	67	498.02	7.03	49.23	72
	Control	✓ 194.42	6.48	56.92	31	✓ 363.71	6.02	43.55	16	✓ 520.78	5.67	51.54	30
	Hybrid	✓ 187.96	5.25	54.62	32	✓ 363.47	5.61	43.55	77	✓ 498.73	5.33	49.23	72
(20, 5)	Queueing	164.03	24.48	46.92	32	344.15	18.77	41.61	67	472.23	13.64	46.41	72
	Control	197.08	7.31	57.69	32	✓ 366.94	6.58	43.87	18	✓ 518.05	5.88	51.28	29
	Hybrid	✓ 184.57	6.13	53.85	32	✓ 363.69	6.05	43.55	77	✓ 495.24	6.08	48.97	72
(10, 10)	Queueing	182.61	7.79	48.46	59	400.81	11.18	45.16	123	✓ 561.79	4.47	50.00	179
	Control	✓ 206.58	6.34	56.15	57	✓ 412.16	5.86	43.23	30	✓ 576.74	5.82	51.54	48
	Hybrid	✓ 198.03	5.49	53.85	59	✓ 412.72	6.47	43.23	172	✓ 548.96	5.89	48.72	179
	IBM	83.03	46.20	26.92	36	155.88	44.98	20.97	70	227.84	65.32	24.36	101
(10, 5)	Queueing	177.44	26.46	46.92	59	372.23	13.87	41.29	123	525.46	15.81	46.15	179
	Control	204.23	6.86	55.38	45	✓ 413.46	6.26	43.55	22	✓ 566.30	5.43	50.51	37
	Hybrid	199.66	6.89	53.85	59	✓ 419.60	6.14	44.19	172	✓ 551.71	5.69	48.97	179

Table 4: Comparing the Schemes for the three workloads. Only the costs with a “✓” next to them are “viable”.

6.2.2 Comparison with IBM method

The most closely related power management mechanism from IBM [15] uses a rather simple feedback control mechanism to determine the frequency and number of servers for the next interval given those of the current interval and the observed utilization/response time, and these two mechanisms are integrated (i.e. $T = t$). It is a purely feedback mechanism and does not use any prediction, and it is not intended to manage server allocation across applications. In the interest of fairness, we should mainly compare results for WL1 (the single application workload in Table 4), and even there we see that while their method gives better energy savings (around 70%), it is not “viable” (i.e. the response times are never meeting the required SLA). Since their method treats response time on a best effort basis, rather than a constraint to obey, energy optimization takes center-stage, leading to much worse response times. As mentioned earlier, the SLA is usually much more important (the revenue earner) to the hosting center. Our schemes are able to meet the SLA in many cases, while still providing reasonable energy savings (around 50%).

6.2.3 Comparing our schemes

Table 4 can be used to compare our three strategies for the three workloads. We mainly focus on the “viable” executions, i.e. those where the average response time is less than 6.6 ms.

As described earlier, the pro-active Queuing approach uses predictions to anticipate workload behavior and steady-state analysis, both of which need larger time granularities for better accuracy. At these larger granularities, the Control theoretic approach does not have any prediction information to optimize for the next interval, using feedback from the last interval that is again coarse-grained. Consequently, we see that the Queuing approach does much better than the reactive Control scheme at large (T, t) granularities.

For example, at (60,60), Queuing is around 10% better in cost than Control for WL1.

To understand the execution characteristics of these experiments at large (T, t) , we give an example execution of Application 3 in WL3 in terms of the number of servers provisioned (m_i), their frequency (f_i) and the corresponding response time (W_i) over time in Figure 5 (b). If we examine the response time behavior of Control (60,60), we see that there is a spike at hour 7. This is because there was a load burst in this hour, which the mechanism could not detect at the end of hour 6 (because there is only feedback). On the other hand, Queuing has advance knowledge of the workload for the next hour, and is able to make better judgments on server allocation. This also reflects on the corresponding frequency that is being set, resulting in the lower cost for Queuing.

When we move to the other extreme of small time granularities (say (10,5)), we see that the inaccuracies of the Queuing strategy significantly impact its viability, and this option does not meet the SLA in any of the 3 workloads. However, by getting very frequent feedback from the system, the reactive Control scheme is able to perform a better job of server allocation and frequency modulation to lower the costs.

The result of inaccuracies of Queuing at small (T, t) is obvious in Figure 5 (a), which shows the time varying behavior for Application 2 in WL3. The Queuing scheme appears to under-estimate the required capacity, thus turning down more servers than Control. This results in a higher response time.

The hybrid scheme that is intended to benefit from the merits of these two schemes, does give a cost between these two at either extremes of time granularities. When we look at the large T and short t ranges (e.g. (60,5)), hybrid benefits from the pro-active mechanism for server allocation from the queuing strategy (it uses the same number of servers as Queuing), and the reactive feedback

from the underlying system to set frequencies based on short term variations to meet the response time SLA.

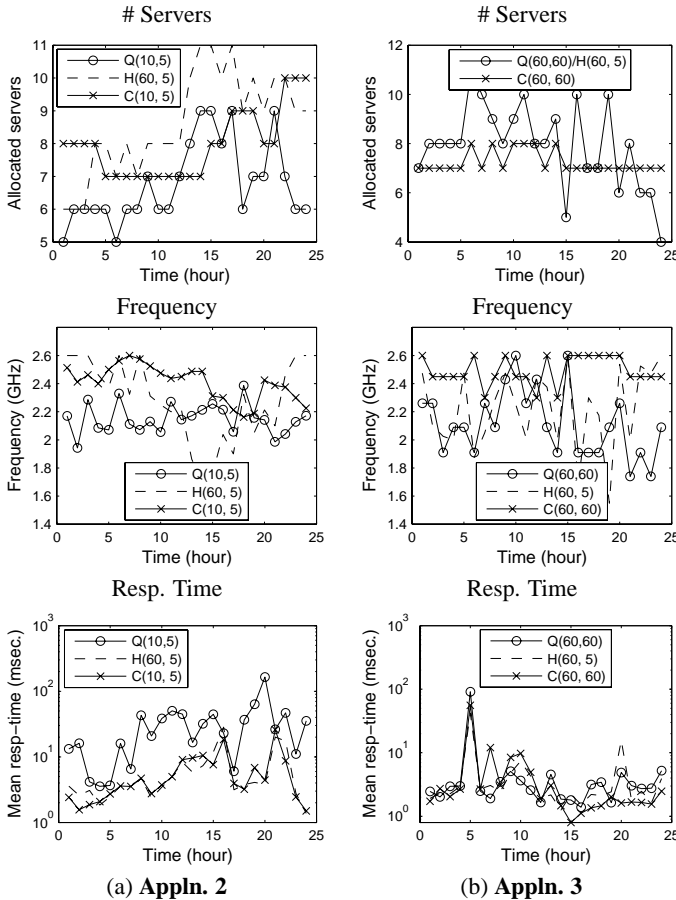


Figure 5: Time Varying Behavior for Appln. 2 and Appln. 3 in WL3

6.3 Varying the Parameters

One would be interested in finding out how these schemes scale as we move to a larger environment with a lot more servers that host more applications. We have conducted an experiment with $M = 285$ servers running $N = 30$ applications. Since it is difficult to procure so many different traces, we take the three traces used earlier and replicate them by using randomized phase differences to synthetically generate workloads. Results for this workload are given in Table 5 for representative (T, t) granularities, viz., (60,60), (60,5) and (10,5). As before, the Queuing approach is more viable at the larger time granularity, and the inaccuracies cause a significant degradation in response times at the finer granularity. In fact, these inaccuracies at fine granularities result in poor choice of server allocation in Hybrid at (10, 5). However, at the (60, 5) granularity, Hybrid is again able to benefit from the pro-active prediction of Queuing for server allocation, and the feedback of Control for frequency modulation, giving the lowest response times.

Note that by modulating parameters of our system model we can target our optimizations at a wide spectrum of systems, operating conditions and cost functions. For instance, setting T to ∞ implies conducting server provisioning once (i.e. a static configuration of the number of servers), and then managing power only with DVS. Similarly, setting t to ∞ implies that only server turn off/on is em-

(T,t)	Schemes	Cost (cents)	RT (ms)	#reboots
(60, 60)	Queueing	$\sqrt{2625.2}$	5.5	275
	Control	$\sqrt{2618.7}$	6.4	215
	Hybrid	$\sqrt{2619.7}$	5.6	275
(60, 5)	Queueing	2465.1	35.4	275
	Control	$\sqrt{2579.6}$	6.5	213
	Hybrid	$\sqrt{2612.3}$	5.4	275
(10, 5)	Queueing	2435.5	73.4	342
	Control	2628.7	7.2	323
	Hybrid	2498.4	30.8	342

Table 5: Results for Workload with 30 applications and 285 servers

ployed for power management. We have also conducted experiments varying the dollar (B_o) and time (T_{reboot}) of reboots (B_o), and the time for application migration ($T_{migrate}$). The reader is referred to [13] for these results.

Further, we can ignore the current objective function, set $\mathcal{F} = f_{max}$, and find the minimum number of servers needed to meet W_i SLA for each application at any time. This is the traditional dynamic server provisioning problem without considering energy consumption. We have conducted such an experiment for WL3, and we find that the number of servers that are needed in all is 29 where the m_i keeps changing over time (at 60 minute granularity using Queuing). This is lower than $11 + 11 + 11 = 33$ servers that would be needed if each of the applications were to be run in isolation, i.e. the server farm is partitioned between the applications to ensure SLA for each. This is because the peak demands across applications are not necessarily coming at the same time. Note that we are able to optimize energy by considering server provisioning and DVS simultaneously. On the other hand, schemes such as [15], rely on a higher level provisioning mechanism after which they can perform their energy optimizations.

7. CONCLUDING REMARKS

This paper has presented the first formalism to the problem of reducing server energy consumption at hosting centers running multiple applications towards the goal of meeting performance based SLAs to client requests. Though prior studies have shown energy savings for server clusters by server turn-offs and DVS, these savings come at a rather expensive cost in terms of violating the performance-based SLAs (which are extremely important to maintain the revenue stream). Further, previous proposals have not considered the cost of server turn-offs, not just in terms of time overheads, but also in the wear-and-tear of components over an extended period of time.

Our solution strategies couple (i) server provisioning for different applications, and (ii) DVS towards enhancing power savings, while still allowing different mechanisms for achieving these two actions, unlike any prior work. We have presented three new solution strategies to this problem. The first is a *pro-active* solution (Queuing) that predicts workload behavior for the near future, and uses this information to conduct non-linear optimization based on steady state queuing analysis. The second is a *reactive* solution (Control), which uses periodic feedback of system execution in a control-theoretic framework to achieve the goals. While Queuing can be a better alternative than Control when the workload behavior for the near future is different from the recent past, the steady state assumptions may not hold over shorter granularities. Feedback Control is preferable at these shorter time granularities, but the information obtained at the last monitoring period may not be very recent at larger time granularities. The Hybrid strategy, on the other hand, can use the predictive information of Queuing at coarse

time granularities for server provisioning, and the feedback of Control at short granularities for DVS. This Hybrid scheme is also well suited for a more practical setting where one would like to perform server provisioning less frequently (due to the high time overhead of bringing up servers or migrating applications), and perform DVS control more frequently. We have demonstrated these ideas using real web server traces, showing how our generic framework can capture a wide spectrum of system configurations, workload behavior, and the targets for optimization. Finally, we would like to mention that an implementation of the proposed schemes strategy is not time-consuming. For instance, in the 285 servers running 30 applications experiment, each server allocation invocation for Hybrid (which is done once in 60 minutes) takes around 30 seconds, and the frequency calculations (done once in 5 minutes) take less than a second, and these would not even need to run on the actual server nodes. By doing this control, we are saving on the average around \$35 per day in electricity for the 285 server system. If we consider a more realistic hosting center with 10 times as many servers, this can work out to a savings of over \$125K per year.

We are currently prototyping this framework on a server cluster. We are also refining our solution strategies further, and evaluating them with a wider-spectrum of workloads.

8. ACKNOWLEDGEMENTS

This research has been funded in part by NSF grants 0325056, 0429500, 0130143 and an IBM Faculty Award.

9. REFERENCES

- [1] Web Caching project. <http://www.ircache.net>.
- [2] Intel Outlines Platform Innovations for More Manageable, Balanced and Secure Enterprise Computing. Intel Press Release, February 2004. <http://www.intel.com/ca/pressroom/2004/0218b.htm>.
- [3] T. Abdelzaher, K. G. Shin, and N. Bhatti. Performance guarantees for Web server end-systems: A control-theoretical approach. *IEEE Transactions on Parallel and Distributed Systems*, 13(1), 2002.
- [4] B. D. O. Anderson and J. B. Moore. *Optimal Control: Linear Quadratic Methods*. Prentice Hall, 1989.
- [5] K. Appleby, S. Fakhouri, L. Fong, G. Goldszmidt, M. Kalantar, S. Krishnakumar, D. Pazel, J. Pershing, and B. Rochwarger. Ocean-SLA Based Management of a Computing Utility. In *Proceedings of the IEEE/IFIP Integrated Network Management*, May 2001.
- [6] P. Bohrer, M. Elnozahy, M. Kistler, C. Lefurgy, C. McDowell, and R. Rajamony. The Case for Power Management in Web Servers. In R. Graybill and R. Melhem, editors, *Power Aware Computing*. Kluwer Academic Publishers, 2002.
- [7] G. Bolch, S. Greiner, H. Meer, and K. S. Trivedi. *Queueing Networks and Markov Chains: Modeling and Performance Evaluation with Computer Science Applications*. John Wiley, New York, 1998.
- [8] E. Carrera, E. Pinheiro, and R. Bianchini. Conserving Disk Energy in Network Servers. In *Proceedings of the 17th International Conference on Supercomputing*, 2003.
- [9] A. Chandra, P. Goyal, and P. Shenoy. Quantifying the Benefits of Resource Multiplexing in On-Demand Data Centers. In *Proceedings of First ACM Workshop on Algorithms and Architectures for Self-Managing Systems*, June 2003.
- [10] A. Chandrakasan and R. W. Brodersen. *Low-Power CMOS Design*. Wiley-IEEE Press, 1998.
- [11] J. Chase, D. Anderson, P. Thakur, and A. Vahdat. Managing Energy and Server Resources in Hosting Centers. In *Proceedings of the 18th Symposium on Operating Systems Principles SOSP'01*, October 2001.
- [12] J. Chase and R. Doyle. Balance of Power: Energy Management for Server Clusters. In *Proceedings of the 8th Workshop on Hot Topics in Operating Systems*, May 2001.
- [13] Y. Chen, A. Das, W. Qin, A. Sivasubramaniam, Q. Wang, and N. Gautam. Managing Server Energy and Operational Costs in Hosting Centers. Technical Report CSE-05-002, The Pennsylvania State University, February 2005.
- [14] J. G. Elerath. Specifying Reliability in the Disk Drive Industry: No More MTBFs. In *Proceedings of the Annual Reliability and Maintainability Symposium*, pages 194–199, 2000.
- [15] M. Elnozahy, M. Kistler, and R. Rajamony. Energy-Efficient Server Clusters. In *Proceedings of the Second Workshop on Power Aware Computing Systems*, February 2002.
- [16] M. Elnozahy, M. Kistler, and R. Rajamony. Energy Conservation Policies for Web Servers. In *Proceedings of the 4th USENIX Symposium on Internet Technologies and Systems*, March 2003.
- [17] K. Flautner, S. Reinhardt, and T. Mudge. Automatic performance setting for dynamic voltage scaling. In *Proceedings of the 7th annual international conference on Mobile computing and networking*, pages 260–271, 2001.
- [18] N. Gandhi, S. Parekh, J. Hellerstein, and D. Tilbury. Feedback Control of a Lotus Notes Server: Modeling and Control Design. In *Proceedings of the American Control Conference*, 2001.
- [19] D. Grunwald, P. Levis, K. I. Farkas, C. B. Morrey III, and M. Neufeld. Policies for Dynamic Clock Scheduling. In *Proceedings of the Symposium on Operating Systems Design and Implementation*, pages 73–86, 2000.
- [20] S. Gurumurthi, A. Sivasubramaniam, M. Kandemir, and H. Franke. DRPM: Dynamic Speed Control for Power Management in Server Class Disks. In *Proceedings of the International Symposium on Computer Architecture*, pages 169–179, 2003.
- [21] D. P. Helmbold, D. E. Long, T. L. Sconyers, and B. Sherrod. Adaptive disk spindown for mobile computers. *Mob. Netw. Appl.*, 5(4):285–297, 2000.
- [22] J. Jones and B. Fonseca. Energy Crisis Pinches Hosting Vendors. <http://iwsun4.infoworld.com/articles/hn/xml/01/01/08/010108hnpower.xml>.
- [23] C. Lefurgy, K. Rajamani, F. Rawson, W. Felter, M. Kistler, and T. W. Kelle. Energy Management for Commercial Servers. *IEEE Computer*, 36(12):39–48, 2003.
- [24] K. Li, R. Kumpf, P. Horton, and T. E. Anderson. Quantitative Analysis of Disk Drive Power Management in Portable Computers. Technical report, University of California at Berkeley, 1993.
- [25] J. R. Lorch and A. J. Smith. Improving Dynamic Voltage Scaling Algorithms with PACE. In *Proceedings of ACM SIGMETRICS*, June 2001.
- [26] C. Lu, T. F. Abdelzaher, J. A. Stankovic, and S. H. Son. A Feedback Control Approach for Guaranteeing Relative Delays in Web Servers. In *Proceedings of the Seventh Real-Time Technology and Applications Symposium*, page 51, 2001.
- [27] C. D. Patel, C. E. Bash, C. Belady, L. Stahl, and D. Sullivan. Computational Fluid Dynamics Modeling of High Compute Density Data Centers to Assure System Inlet Air Specifications. In *Proceedings of the Pacific Rim ASME International Electronic Packaging Technical Conference and Exhibition (IPACK)*, 2001.
- [28] T. Pering, T. Burd, and R. Brodersen. The simulation and evaluation of dynamic voltage scaling algorithms. In *Proceedings of the 1998 international symposium on Low power electronics and design*, pages 76–81, 1998.
- [29] E. Pinheiro, R. Bianchini, E. V. Carrera, and T. Heath. Load Balancing and Unbalancing for Power and Performance in Cluster-Based Systems. In *Proceedings of the Workshop on Compilers and Operating Systems for Low Power*, September 2001.
- [30] P. Pradhan, R. Tewari, S. Sahu, A. Chandra, and P. Shenoy. An Observation-based Approach Towards Self-managing Web Servers. In *Proceedings of ACM/IEEE Intl Workshop on Quality of Service*, May 2002.
- [31] S. Ranjan, J. Rolia, H. Fu, and E. Knightly. QoS-Driven Server Migration for Internet Data Centers. In *Proceedings of International Workshop on QoS*, May 2002.
- [32] K. Roy and S. Prasad. *Low-Power CMOS VLSI Circuit Design*. John Wiley and Sons, New York, 2000.
- [33] V. Sharma, A. Thomas, T. Abdelzaher, and K. Skadron. Power-aware QoS Management in Web Servers. In *Proceedings of the Real-Time Systems Symposium*, December 2003.
- [34] K. Shen, H. Tang, T. Yang, and L. Chu. Integrated resource management for cluster-based Internet services. *SIGOPS Oper. Syst. Rev.*, 36(11):225–238, 2002.
- [35] D. S. Stoffer and R. H. Shumway. *Time Series Analysis and Its Applications*. Springer Verlag, New York, 2000.
- [36] B. Urganonkar and P. Shenoy. Cataclysm: Handling Extreme Overloads in Internet Services. In *Proceedings of ACM Principles of Distributed Computing*, July 2004.
- [37] M. Weiser, B. Welch, A. J. Demers, and S. Shenker. Scheduling for Reduced CPU Energy. In *Proceedings of the Symposium on Operating Systems Design and Implementation*, pages 13–23, 1994.
- [38] Q. Zhang, E. Smirni, and G. Ciardo. Profit-driven Service Differentiation in Transient Environments. In *Proceedings of the 11th IEEE/ACM International Symposium on Modeling, Analysis and Simulation of Computer Telecommunications Systems*, 2003.