On Evaluating Request-Distribution Schemes for Saving Energy in Server Clusters

Karthick Rajamani
karthick@us.ibm.com

Charles Lefurgy
lefurgy@us.ibm.com

IBM Austin Research Lab

Abstract

Power-performance optimization is a relatively new problem area particularly in the context of server clusters. Power-aware request distribution is a method of scheduling service requests among servers in a cluster so that energy consumption is minimized, while maintaining a particular level of performance. Energy efficiency is obtained by powering-down some servers when the desired quality of service can be met with fewer servers. We have found that it is critical to take into account the system and workload factors during both the design and the evaluation of such request distribution schemes. We identify the key system and workload factors that impact such policies and their effectiveness in saving energy. We measure a web cluster running an industry-standard commercial web workload to demonstrate that understanding this system-workload context is critical to performing valid evaluations and even for improving the energy-saving schemes.

1 Introduction

Energy efficiency for servers has recently become an important concern for technical, environmental, and financial reasons [1]. Both products [2, 3, 4] and research [5, 6, 7] are beginning to address the problem of reducing energy consumption in servers. Energy conservation policies for server clusters use variations in workload to conserve energy by reducing resource consumption when the load is low. When using such policies, both the system and workload characteristics affect the energy savings that can be obtained. Previous research has not considered these characteristics in detail. In this paper, we present a systematic study of the system and workload characteristics that affect the design and evaluation of request distribution schemes targeting energy savings in server clusters.

With the growing trend of dynamic content web sites, we employ *Power-Aware Request Distribution* (PARD) at the load-balancing front-end of a cluster serving dynamic web workloads. Energy management is performed for the server tier which produces the dynamic content. The static (image) service and the database back-end are hosted on separate servers that are not energy-managed. We generate our experimental workloads from the industry-standard TPC-W benchmark [8] and apply traces from real site logs to produce varying load profiles. This allows us to capture the dynamic workload behavior from TPC-W while using realistic load variations to test the effectiveness of energy-conserving schemes. We use a custom-built, energy-efficient server cluster in our evaluations to examine and validate our theories.

In this paper, we identify the following key system and workload factors that most significantly affect energy efficiency. The primary system factors are the *cluster unit* and its capacity, startup delay, shutdown delay, and the ability (or lack thereof) to migrate connections between servers. The primary workload factors are the load profile with a specific shape and peak, the rate of change in load in relation to the current load, and the *workload unit*. Together, these factors make up the *system-workload context*. We show the interaction between these factors and how a given system-workload context affects the design and performance of a PARD scheme. Our key contributions are as follows:

- We identify the key factors in the system-workload context that impact energy saving policies. We show how to use the knowledge of these factors to derive better energy estimates for PARD policies.
- We provide a novel method for generating workloads which meet the need for industry-standard benchmarks as well as the load variation required for energy-saving studies.
- We analyze the energy savings for a set of PARD policies using our *on-off* model with the system-workload context. We verify them with energy measurements on a real cluster with dynamic web workloads.

All the PARD schemes we consider save energy by turning off servers that are idle. In a survey of idle server power, Chase et al. [5] found that conventional servers used at least 60% of their peak power when at idle (with the IA-32 processor calling HALT in the idle loop of the operating system). In fact, all but one system used over 74% of their peak power at idle. This idle power consumption is due to components that are not power-managed like memory and sometimes disks and inefficiencies of power supplies. Thus, power-management by just voltage-scaling processors will miss saving significant energy when the system is idle. In our studies, we turn-off servers to obtain the maximum energy savings at idle. Some operating systems have a hibernation mode that uses very little power. This could be more energy-efficient than full reboots of the server after short duration idle periods. We do not evaluate this in our study as our infrastructure does not support this feature.

In our studies, we have chosen to use web workloads in which all characteristics (maximum load, maximum rate of change in load, etc.) are known a priori. We assume a well-provisioned cluster that operates during normal usage conditions without unexpected load increases. Real clusters could experience overload due to equipment failures or unexpected load spurts. Here, we focus on the impact of the system-workload context on request distribution techniques; the design of request distribution schemes to handle unplanned load is beyond the scope of this paper.

The organization of the paper is as follows. In section 2, we present the related work from previous studies. In section 3 we define the system-workload context and describe its importance in understanding the results of energy optimization studies. Section 4 describes our basic request distribution strategy, our system environment and the workloads used. We present our experimental results and analysis in section 5. Finally, we conclude the paper and present some of our ideas for future work in section 6.

2 Related Work

Energy management policies that dynamically resize the cluster in response to workload variation are just beginning to be studied. Pinheiro et al. aggregate the demands made on CPU, network, and disk in the cluster and use this to estimate the number of servers to keep powered-on [7]. The system administrator tunes the system by setting an elapsed time (the time to wait between reconfigurations to let the system settle) and a degradation percent. Chase et al. estimate the impact of cluster resource availability on workload throughput using economic theory [5]. They suggest that a bound on cluster energy savings can be estimated by the variability in the workload. In our work, we examine the interaction between workload variability, system factors, PARD strategies and energy consumption. Elnozahy et al. [6] have studied the relationship between policies that use dynamic voltage scaling and those that turn servers completely off. They found that a policy that both resized the cluster and dynamically varied the voltage (and frequency) of the servers achieved that best energy savings. However, a simple policy that turns off servers when not required is found quite competitive with the more complex policies.

In contrast to these previous studies, our study does not focus on finding the optimal energy management policy. Some of these prior studies lack rigorous explanation of their experimental setup and results which makes comparison between the studies difficult. For example, it is often not clear if the peak of the workload has been correctly sized for the number of servers in the cluster. In this paper, we provide a framework for doing such experiments and list the critical system and workload parameters that are necessary to put the studies in context. We validate our framework by measuring energy and performance on a real cluster. Another point of differentiation is that we are the only study to use a transaction-oriented web workload that is based on an industry-standard benchmark - TPC-W. We describe how we scale our workloads which use the time-varying load from two real web sites. In addition, our experiments are done on a cluster of blade servers that have been designed for energy-efficiency.

While resizing clusters to save energy is a new area of research, there have been many previous studies on resizing the resources of a single server to save energy while maintaining a performance goal. Some examples of these studies include adjusting the performance of the processor using voltage scaling [9] [10], resizing queues in the microprocessor [11], resizing the cache by powering-down unused cache lines [12] [13], and putting idle memory banks into a low-power mode[14].

Energy and performance optimizations in web servers have been widely studied. Bohrer et al. [1] report on the opportunity to use voltage-scaling processors in web servers. Elnozahy et al. [15] use request batching as a method to improve energy-efficiency during low-intensity workloads that would otherwise prevent servers from being turned off. Locality-Aware Request Distribution [16] was initially conceived of as a way to improve performance of web servers by selecting a web server that has the request cached in memory (instead of on disk). Although energy management was not the focus of that study, it is likely that fetching a cached copy of the document will save energy as well. Oceano [17] is a system managing an e-business computing utility. It dynamically assigns servers from a common pool to to multiple customers in response to changing workload to meet quality of service constraints. It would be straight-forward to augment the server assignment with PARD-like policies to reduce energy consumption whenever customers underutilize their allocated servers.

3 The System-Workload Context

The problem of power-aware request distribution can be characterized as minimizing cluster resource utilization for a particular workload, while meeting given quality-of-service (QoS) constraints. Our focus is on those resources whose consumption affects the variable energy in the system. The different dimensions of the problem-space are:

- the energy consumed
- the QoS
- the system characteristics
- the workload characteristics

Every point in this four-dimensional space has an associated implication for the energy consumed by the cluster, the performance provided, cluster design decisions, and the nature of workloads that can be serviced. The system and workload characteristics, together, define the space of operation for the trade-offs between energy savings and QoS. We call this the *system-workload context*. Studies that report energy savings and QoS without defining the system-workload context can give only an incomplete and potentially inaccurate picture of the value of a PARD scheme.

Furthermore, while energy and QoS are the measures of interest to request-distribution strategies, the indirect impact of the system and workload on both energy and QoS makes understanding their roles crucial to both the design and evaluation of energy-saving request distribution schemes. In this section, we identify the critical system-workload factors that impact the efficacy of PARD schemes.

3.1 The *On-Off* Model

We use a simple *On-Off* model for estimating the variable energy consumption of the cluster. We assume that the power of a cluster resource (in our case, a server) is equal to its peak power when it is on and zero when it is off. This assumption lets us compute the energy savings for any duration based on just the average number of resources that are active during that period. For our PARD schemes, in which complete servers are turned on or off, this is a valid assumption. We believe that resources with more power states than just on and off can be considered as multiple units with different on-level energy consumption and can be studied with the same model. Resources with a huge variation in the on-level energy consumption, i.e. with a dynamic energy range that is a significant fraction of the off-to-peak energy change, would need to be approximated with multiple resource units each corresponding to a small range in the energy variation, to work under this model.

We define the *utilization ratio* as

$$utilization\,ratio = \frac{average\ active\ resources}{total\ resources} \qquad (1)$$

A utilization ratio of 1 corresponds to a system that always runs at peak capacity while a utilization ratio of 0 corresponds to a system that is completely idle. The fraction of energy saved is:

$$energy\ saved = 1 - utilization\,ratio \qquad (2)$$

Though, this model is simple, we will see that it is quite effective in predicting the energy consumption provided we understand the implications of the system-workload context on the fraction of resources used and, thus, on the energy consumption of the system.

3.2 Key System Characteristics

Following are the key system characteristics that affect the impact of PARD schemes:

Cluster unit: It is the smallest unit of resource that can be added/removed to increase/decrease energy consumption of the cluster. Its energy cost and performance are key to any PARD scheme. Smaller the size of the cluster unit (in terms of its energy cost), the finer the adjustments that can be made for power-performance trade-offs. In this paper, we assume the cluster unit is a single server.

Immunity to overload: The *capacity* of the cluster unit is the maximum load seen by any cluster unit after which its performance becomes unacceptably low. As PARD schemes vary the resources committed to a service, overload situations can occur even when the full cluster resources are sufficient to handle the increased load. Thus, it is vital to know the capacity of a cluster unit to both detect and avoid overload which is important both for design and evaluation of PARD schemes. In this paper, we measure *capacity* in terms of the number of active client connections to the cluster unit.

System energy consumption: In addition to the energy consumed by a cluster unit, the base energy consumption of the system (minimum system configuration) and the peak energy consumption need to be specified to establish the variability of system energy which would be the domain for PARD-based energy conservation.

System support for PARD scheme implementations and their cost: The following are some important system capabilities that affect the design and impact of PARD schemes.

Ability to turn on cluster units: The *startup delay* is the time to bring a cluster unit online - includes time to power it up, boot the OS and start services, and, if significant, the cost of bringing it to a certain performance level (say, pre-load data into cache/memory). Given a certain minimum required QoS, the startup delay affects the number of cluster units that need to be running to handle spikes in load.

Ability to turn off cluster units: The *shutdown delay* corresponds to the period between removing a cluster unit from service and actually turning it off. During this period the cluster unit would continue to consume energy while providing no service. Larger shutdown delays (combined with the startup delays) could limit the ability of PARD to react to smaller and less predictable changes in load.

Independence of cluster units: When cluster units turn on and off, the less impact they have on the other running cluster units, the less complex and more flexible the PARD schemes can be. For example, if all data is placed on a remote file system (for example, NFS file systems) then any cluster unit's ability to service requests is independent of the status of other cluster units. On the other hand, if data is partitioned between local disks, clusters units cannot be shut off without cutting off access to their local data or before re-partitioning the local data.

Ability to migrate service requests: The ability to transfer or terminate a request/connection to a cluster unit could also impact the design and effect of PARD schemes. The impact of this is closely related to the duration of service for each request/connection. If the duration of the request is long and the system does not provide the ability to migrate the service to another cluster unit, then the servicing cluster unit needs to be kept online for the full duration of the request or the connection as long as there is activity on it.

3.3 Key Workload Characteristics

Following are the key workload characteristics that affect the impact of the PARD schemes:

Workload unit: Analogous to the cluster unit, the workload unit is the minimal schedulable service request. Often, the request can have very different impact on the server load depending on the specific parameters of the request, current resource consumption of the server, and other factors. While this makes determining the impact of each request quite difficult, it is important to have a good understanding of the cost for a request, at least of the average cost. In this paper, we use a single client connection as the basic workload unit.

Load Ratio: We define *load ratio* for the workload as

$$load\ ratio = \frac{average\ load}{peak\ load} \qquad (3)$$

Load ratio of 1 corresponds to a workload with a fixed, constant load affording little scope for PARD-based energy savings. Given the instantaneous load and a required minimum quality of service for the workload, there is a certain minimal number of cluster units (resources) that need to be committed to that service. If we assume a linear relation between load and resource utilization (usually valid under regular operation conditions), for a workload scaled to the system capacity the load ratio would correspond to the minimal *utilization ratio* for the system. With energy consumption proportional to resources used, the load ratio provides a useful upper bound of $\{1 - load\ ratio\}$ for the energy savings.

Rate of change in load in relation to the current load: Given a certain QoS constraint the current load determines the amount of resources required to serve it. Any increase in load has to be met with the same resources until new resources can be brought online. Hence, given a certain delay in bringing up a new server, the maximum rate of change in load that can be tolerated is also determined by the current load. Conversely, both the load profile (load versus time) and the rate of change of the load profile (rate versus time and load) are required to characterize a workload for energy-savings studies.

3.4 Interplay of System-Workload Characteristics

In this section, we illustrate the interactions of the parameters and characteristics identified in the previous section and their impact on energy consumption. First, we show the impact of the workload characteristics and the cluster unit capacity on the maximum achievable energy savings. Next, we show the impact of the startup and shutdown delays.

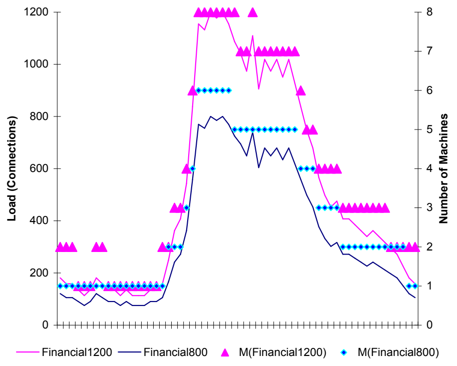

Figure 1: Same workload, different scales

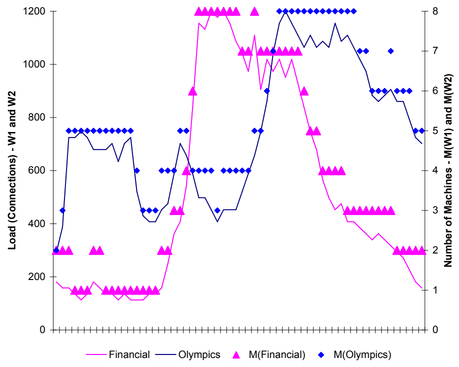

Figure 2: Different workloads, same peak

Figures 1 and 2 depict workloads (load vs time) derived from the logs of a day's activity at two web sites. Figure 1 shows the effect of scaling the same workload by different factors. The system consists of 8 servers each capable of sustaining 150 connections (workload unit = 1 connection, and capacity **C** = 150 connections) at the required QoS for a total capacity of 1200 concurrent connections. The curve Financial1200 corresponds to the workload scaled to the peak while Financial800 is the same workload scaled to 800 connections. M(Financial1200) and M(Financial800) correspond to the number of active servers required to serve the workload at any given instant. The maximum expected energy savings is $\{1 - utilization\ ratio\}$, which is 0.51 for Financial1200 and 0.67 for Financial800. We get $\{1 - load\ ratio\}$ numbers of 0.57 and 0.73 for Financial1200 and Financial800, respectively which are quite close to the energy estimates from the utilization ratios. We also see that Financial800 has a larger estimated energy savings than Financial1200 showing one could get optimistic savings when the evaluation workload is not scaled to the capacity of the cluster.

Figure 2 illustrates the usefulness of the load ratio with two different workload profiles both scaled to 1200 connections at peak. The energy savings computed from the load ratios for Financial1200 and Olympics1200 are 0.57 and 0.36,

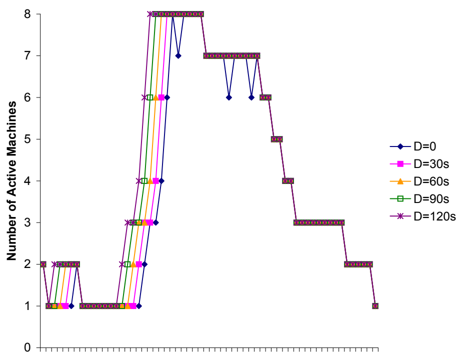

Figure 3: Effect of Startup Delay

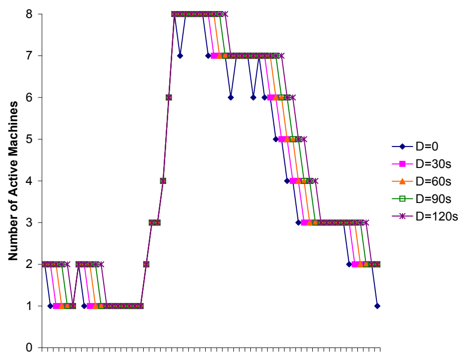

Figure 4: Effect of Shutdown Delay

which are marginally higher than the savings computed from the utilization ratios - 0.51 and 0.33. The examples illustrate that energy savings varies with both the peak and shape of the load curve. They also illustrate that the load ratio can provide a good upper bound on the maximum energy savings when given just the knowledge of the workload.

Next we examine the effect of the system delays on energy savings. Let *D* represent the startup delay; *N*, the number of powered-on machines; *C*, the capacity of each machine; *L*, the instantaneous load on the system; and, *S*, the rate of change/slope of the load on the system. Then, to meet the QoS constraint, we need the following relation to be maintained between the above quantities:

$$L + SD \leq N * C \tag{4}$$

For a system with specific machine capacity *C*, at any given load (*L* and *S*), a higher value of *D* implies a higher value for *N* i.e. a higher startup delay requires more active machines, in general, leading to higher energy consumption. For a given load profile, L_t (i.e. *L* and *S* will be known for all instants of time), one can compute the minimal N_t such that the above inequality is satisfied for all time t. This curve would then correspond to the minimum energy curves for the given load profile and system parameters *D* and *C*. Figure 3 shows N_t curves for different startup delays for the Financial1200 profile from figure 1.

While the effect of startup delay on energy consumption is indirect through its effect on the QoS constraint (which is met by never having more than *C* connections per machine), the shutdown delay has a more direct effect. At any given time t, if *N* machines are active and *J* of them can be shutdown, a shutdown delay of *D* implies that there will be excess energy consumption proportional to $J * D$ for this particular instance of the shutdown delay. Figure 4 shows the N_t curves for different shutdown delays for the Financial1200 profile from figure 1.

4 Experimental Platform

In this section, we describe the environment in which we ran our experiments. The three major components of this environment are the Super Dense Server that host the web site, the request distribution mechanism, and our modified TPC-W workload.

4.1 System - Super Dense Servers

Our experiments are done on a prototype Super-Dense Server (SDS) cluster that has been designed for energy-efficient web serving [18]. In a previous study we found that the SDS cluster used half the energy of a conventional 1.2GHz Pentium III Xeon Intel rack server while providing similar peak performance. Measuring energy-saving software techniques on hardware that is inefficient may tend to magnify the benefit of the software technique. Therefore, we have chosen the SDS cluster as the platform for this study.

The cluster prototype consists of 8 custom-built low-energy *server blades*, a Ziatech *management blade*, and a Zynx *network switch blade*. All of the blades plug into a CompactPCI [19] chassis that in turn supplies power and cooling. Each blade has a 500 Mhz Intel Pentium III processor with a 256K L2 cache, 256 MB of DDR-266 SDRAM, a USB port, two 100 Mb/sec Ethernet connections, and a Silicon Integrated Systems 635 chipset that combines the function of both north and south bridges onto a single chip. We only use 1 100 Mb/s Ethernet connection on each server in this study. Unlike similar designs [2, 3, 4], our server blades do not contain a disk or KVM connections (Keyboard, video and mouse). In addition, there is no graphics co-processor. Eliminating these items reduces the energy consumption of the blade. An H8 8-bit microcontroller on each server blade provides status and management functions, including the ability to power on/off all devices on the server (besides the H8 itself).

Due to a manufacturing error, our prototype server blades do not properly receive interrupts from external devices on the PCI bus. This problem was overcome by modifying the Linux kernel and network device driver to poll for interrupts. However, this could potentially reduce the gap between the energy consumption of an idle blade and that of a heavily loaded blade. In our runs, a blade's power consumption varies between 13.3 W to 14.8 W depending on activity.

The management blade includes a 20 GB laptop disk. This disk provides the system image for the server blades via NFS. The management blade can power-on and power-off the server blades by sending commands to the H8 microcontrollers via the I2C network across the backplane. The switch blade takes a 2 Gb/s connection from outside the cluster and distributes it over the CompactPCI backplane to the server blades at 100 Mb/s.

The server blades use DHCP to boot and download a Linux kernel from the management blade. Since each blade has known, static hardware, the kernel is configured with the right network driver and no other device drivers. This removes boot-time delays due to device probing. Booting takes 20 seconds and starting up web services takes an additional 10 seconds. Since the blade is diskless and has no non-volatile state it can be shutdown and powered-off in under 1 second.

Energy is measured by attaching sense resistors on the power plug for the chassis so that every component in the system is accounted for, except the fans which use a separate external power supply. A National Instruments data acquisition board in a stand-alone PC system measures the current and voltage from the wall power at 10 K samples per second, converts this to energy, and adds it to the count of total energy used during the experiment. The time-stamped energy counts are sent to a user-level daemon, *Pard*, which also controls the request-distribution infrastructure that is discussed next.

4.2 Request Distribution Infrastructure

The two primary tasks of any power-aware request distribution strategy are to distribute incoming requests among active servers and set the power-state of each server.

Request distribution is performed using the Linux Virtual Server [20] (LVS) software. LVS is run in Direct Routing mode which modifies the IP headers of the reply packets so that servers can respond directly to the clients without sending packets back through the load-balancer. Since the load-balancer only sees the short client requests, its network load is reduced. LVS supports many policies to distribute the connections. We use a modification of the Least Connections (LC) Policy that we call Least Active Connections (LAC). LC balances the load between servers using a weighted formula for load involving the number of active and inactive connections with a higher weight for active connections. It directs new incoming connections to the server with the lowest value for this formula. LAC uses just the active connections for making this decision for two reasons: (1) An active connection has significantly greater load than an inactive connection and even more so for dynamic workloads that we use, and (2) LVS does not see servers close inactive connections, so they remain in the LVS connections table (till they timeout) causing incorrect estimates for the load on

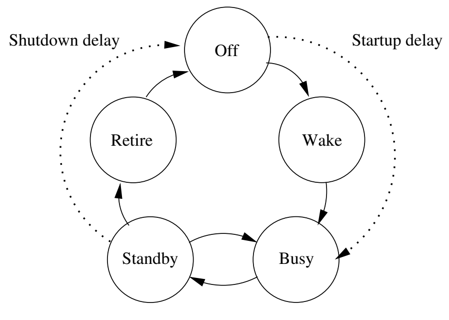

Figure 5: Server state transition diagram

the servers. Additional minor modifications help in removing any remaining temporary imbalances in connections (and load) between servers. The modified LAC tries to ensure that all active servers see the same number of active connections. At any instant the server with the least load receives the new incoming connection.

Managing server power-states with our PARD algorithms is done by a user-level daemon, *Pard*, which also runs on the load-balancer. LVS has been modified to interact with it. Besides managing power, Pard also collects energy measurements, resource usage statistics and operational information from the server machines and other support machines. This data is correlated and logged to produce data for our studies. Pard obtains information from LVS that includes the number of connections on each servers. It uses this information and an internal algorithm to decide on the number of servers to be kept powered-on. If the number of servers needs to be increased, it starts up the necessary number of additional servers (sends the appropriate signal to the management blade to wake up servers, boot them, start server daemons, etc.), waits until they are ready to serve incoming connections and then directs LVS to begin scheduling connections to them. If the number of servers needs to be reduced, Pard selects servers to power-down and directs LVS to stop using them. Once all the connections on these servers have closed, Pard directs the management blade to power them off.

Figure 5 shows the state transition diagram for servers under Pard's control. Pard initially places a server in the *Off* state. When Pard activates a server, it moves it to *Wakeup*. The server remains in *Wakeup* until its boot sequence is complete and all server daemons necessary for servicing requests are running. Pard monitors the progress of the startup of server daemons detecting when they are actually ready to service requests. At this point, Pard moves the server to the *Busy* state and signals LVS to start using the server for incoming connections. The *Startup delay*, discussed in the earlier section corresponds to the time between a server's startup until it moves to the *Busy* state. When Pard decides that a server should be turned off, it places the server in the *Standby* state and directs LVS not to use it. LVS will not send any new connections to that server anymore. Pard then waits until all the existing connections on the server close. Then the server is moved to the *Retire* state. Any required shutdown processes are completed in the *Retire* phase and then Pard signals the server to switch off. The *Shutdown delay*, discussed previously, corresponds to the time between a server's *Standby* phase and when its switched off.

The basic algorithm to determine the required number of servers is called Simple Threshold (ST). Pard obtains the load (in terms of the number of connections) at any instant, L_t, from LVS and feeds this value to ST. ST uses a simple threshold parameter *T* to determine the number of servers necessary as given in Equation 5.

$$N_t \quad = \quad L_t/T \tag{5}$$

As mentioned before, the capacity **C** of servers is measured in terms of the number of active connections for a particular workload. **C** is determined as the maximum number of sustainable active connections on a single server while meeting the QoS constraint. Thus, to meet the QoS constraint no server should be serving more than **C** connections. Since, LAC distributes the load evenly among the active servers this amounts to the average load per server not exceeding **C** connections. For a given startup delay *D*, this places a lower bound on the number of servers required to be powered-on at time t, N_t, as outlined in Equation 6. Here, $\hat{L}_{t,t+D}$ represents the maximum value of the load from t to $t + D$.

$$N_t \quad \geq \quad \hat{L}_{t,t+D}/C \tag{6}$$

Given full knowledge of the system and workload, the responsibility for enforcing the QoS constraints comes down to choosing the appropriate value for the threshold parameter, *T*:

$$\begin{aligned} \hat{L}_{t,t+D}/C &\leq L_t/T \qquad (7)\\ T &= \min_t(L_t/(\hat{L}_{t,t+D}/C)) \end{aligned}$$

For a given profile a higher startup delay then could imply a lower threshold to maintain the same QoS. This can be illustrated by a simple example. Consider, a workload profile with a portion where the load is increasing at the rate of 10 connections/second. Also consider, two startup delays 10 seconds and 20 seconds. At time t, if the load is 100 connections, at t+10 it would be 200 connections and at t+20 it would be 300 connections. Let C for the system be 100 connections. Then at least 2 servers need to be powered-on at t for a 10 second delay and 3 servers for a 20 second startup delay, as otherwise, the load on the active servers would exceed 100 connections. This then requires T to be 50 ($(L_t = 100/2)$ for the 10 second delay and 33 ($(L_t = 100/3)$ for the 20 second delay.

4.3 Workload - Modified TPC-W

We use the specifications for the e-commerce benchmark, TPC-W [8], to build our workload. This provides a dynamic web workload that has characteristics certified as realistic, by the Transaction Processing Performance Council. TPC-W models an online retail book store. The specification gives three different workloads - browsing, shopping, and ordering - which have different ratios of browsing (read) to ordering (write) interactions. We use the shopping workload, which is deemed to be the most representative workload by the council, in all our experiments. Details on the performance characteristics of TPC-W can be found in a paper by Amza et al. [21].

We implement all three tiers for the TPC-W architecture. The database backend consists of MySQL [22] with our own implementation of query result caching to scale it up. We use Apache [23] for our web server with the PHP [24] module for dynamic content generation including communication with the database backend. We scale the web server tier by using PHP script caching from Alternate PHP Cache [25]. We use a separate machine with the Tux web server [26] for serving the images required in TPC-W. The site users are emulated by client programs running on separate machines which open a connection per emulated user to the web server machines. We use the Super Dense Server (section 4.1) for running our Apache web servers with the PHP module. Our machine configuration for the experiments is shown in table 1.

TPC-W is a peak performance benchmark, where for the specific scale of implementation, the metric of importance is the maximum achieved site throughput (in Web Interactions per Second or WIPS) while meeting the 90%-ile response time restrictions. We use this as our QoS constraint. For an energy-savings oriented study where the focus is on resource minimization when demand has been lowered, benchmarks that place peak demand throughout the measurement period are useless. To address this issue, we use a varying load profile on top of the TPC-W workload. Each client program is fed a varying load profile that consists of the number of emulated site users for each time interval for the course of the run. The clients maintain the number of connections to the site according to this profile, thereby generating a varying load to the site for the duration of the run. This approach generates meaningful and interesting workloads for our experiments that have both realistic dynamic content requests and load that varies over time. Interested readers can contact the authors for more details and for the profiles used in our experiments.

We use the web logs for one day from two different web sites - a major financial organization and the 1998 Winter Olympics - to generate our load profiles. The 24 hour log is compressed to a 30 minute profile. Requests received in each 48 minute interval in the original log are counted for 1 minute in the compressed profile. The load profile thus

Table 1: Machine Arrangement for Experiments

Role	**Configuration**	**Software**
Database Server	IBM xSeries 330 - 2 1.26GHz Pentium III, 4G SDRAM	MySQL v 3.23.49a
Web Server	8 SDS blades	Apache v 1.3.23 with PHP 4.1.2
Image Server	IBM xSeries 330 - 2 1.26 GHz Pentium III, 4G SDRAM	Tux v 2.2
Clients	2 IBM xSeries 330: using 1.26GHz Pentium III	C Client program

Table 2: Parameter values used across experiments

Parameter	**Symbol**	**Value**
Capacity of one server	C	140 connections
Total number of servers	–	8
Startup Delay	D	30 seconds
Shutdown Delay	–	0

obtained is then scaled to whatever peak number of connections is required for a particular run. Thus, if the peak of the compressed profile is 1000 requests and the peak for the run desired is 1200 connections, each point on the compressed profile would be multiplied by 1.2 to obtain the desired load profile. Figure 2 shows the two load profile curves scaled to a peak of 1200 connections. We will refer to the two workloads throughout the rest of the paper as Financial and Olympics.

4.4 System-Workload Parameters

The cluster unit for our experiments is an SDS blade. We use a total of 8 blades in our experiments. The non-variable, base energy of our system is 87 Watts. This is the energy for the system management blade, network switch blade and the power inefficiencies in the power supply for the chassis. A fully configured chassis similar to the one we use would accommodate 24 blades, thereby, amortizing this cost over a much larger number of blades. An active blade consumes between 13.3W to 14.8W depending on load. In the rest of the paper, all energy figures are for the variable energy in the system which is over and above this fixed cost. This is the energy whose conservation is addressed by the PARD energy-saving schemes.

The parameter values for our experiments are in table 2. The capacity, C, determined as 140 connections is the number of connections on a server for its peak TPC-W throughput, while meeting the 90%-ile response time thresholds given in the TPC-W specifications. The startup delay of a server blade typically varies between 20 to 30 seconds. To have repeatable experiments, we set Pard to wait 30 seconds after server startup before it signals LVS to begin using the server. Servers have essentially no shutdown delay. As soon as Pard detects that a server has zero active connections, the server is moved from *Standby* to *Retire* after which it powers off in less than a second. Our setup does not have the ability to migrate connections from one server to another inbetween requests (discussed in section 3.2). So, even when Pard decides that a server should be turned off, the server would continue to service requests on its existing connections while it is in the *standby* state. With TPC-W (with a maximum session time of an hour), connections could remain active for up to an hour in *standby* causing an inordinate delay in shutting down the server. We address this by having the client use non-persistent connections, i.e. the clients use a new connection for every dynamic content request from SDS. This reduces our peak throughput a bit (from just above 150 connections to 140 connections), but results in a better environment for power-aware server studies.

5 Experiments and Analysis

In this section, we discuss the performance of different request distribution schemes and the energy savings measured. First we examine the performance of Simple Threshold (ST). Then, we consider two additional schemes with potentially larger energy savings: (a) Spare servers and (b) History-based. Their description, analytical evaluation and experimental verification are provided together in this section.

5.1 Simple Threshold Results

In the ST approach, the threshold T is computed from Equation 7 as the highest value that would meet the QoS constraint of having at most **C** connections per server at any time during the run. Any higher value for T will result in failure to meet the QoS constraint and a lower value will result in lower energy savings. Thus, understanding the system-workload context captured in Equation 7 is the key to obtaining the best possible energy savings for ST. The threshold values determined in this fashion are 100 for Financial and 89 for Olympics. An online variation of ST could adjust T as load knowledge becomes incrementally available - the details are outside the scope of this paper.

Figures 6 and 7 show the number of active blades and the energy consumption for Financial (area under the power curve shown in the figure), respectively, for the ST approach outlined in section 4.2. The *required* curve in figure 6 shows the minimum number of blades required to service the load at any time without violating the QoS restraint (number of connections per blade cannot exceed 140). The *active* curve shows the expected number of blades running under ST

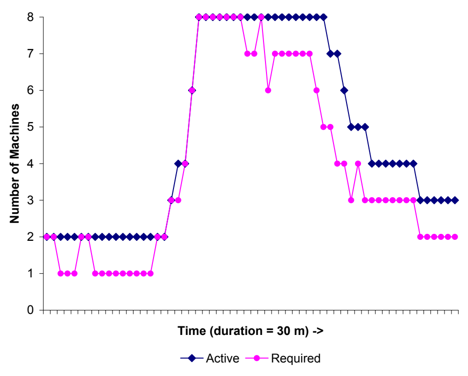

Figure 6: Predicted servers (Financial)

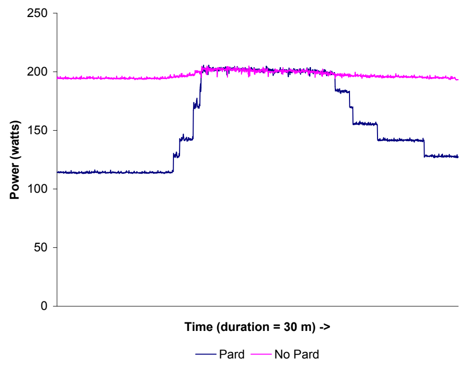

Figure 7: Energy consumption (Financial)

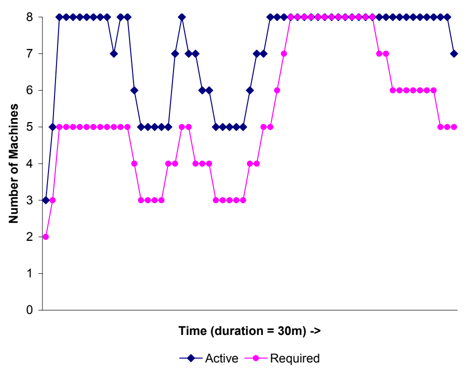

Figure 8: Predicted servers (Olympics)

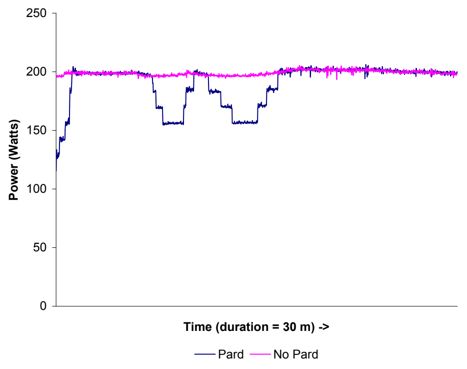

Figure 9: Energy consumption (Olympics)

(from 5). The average number of blades active is 4.76 indicating a 40% savings in blade energy, as opposed to the 51% savings indicated by the *required* curve. Figure 7 shows the actual energy consumption for both the ST Pard scheme and when running the workload against all 8 server blades (*No Pard* curve). Dividing the area of the *Pard* curve by the area of the *No Pard* curve we get 36.7% as the actual energy savings - very close to 40% indicated by the average number of active blades.

Figures 8 and 9 show the corresponding details for Olympics. The *required* machines curve indicates that 32% of energy can be saved, however, 7.2 blades are active under ST on the average (implying 10% savings). The actual energy savings (ratio of the area of the *Pard* curve to the area of the *No Pard* curve) is 7.1%.

The discrepancy between the measured savings and the estimated savings (from *active* curve) is because machines that are determined as not required by PARD (not counted as *active* in the estimation) cannot be switched off till all active requests on their connections have been served and the connections closed. They remain in the *standby* state for a while (longer with dynamic content responses) before going to *retire* and then *off* (refer figure 5). Our energy models do not account for this extra energy consumption. Some additional discrepancy also arises from the *on-off* model assuming a fixed energy consumption for an active blade while actual energy consumption varies a little with load.

5.2 New PARD Schemes

The performance of Simple Threshold falls short of the maximum achievable energy savings. Here, we use our knowledge of the system-workload context to design new PARD schemes with better energy savings. For each approach, we compare the estimates for energy savings using our model and the system-workload context with actual measurements on our experimental platform.

5.2.1 Spare Servers

In order to handle spikes, the only mechanism in ST is to have a low enough threshold such that enough servers would be running when a spike occurs to handle it while maintaining the required QoS. However, this results in excess servers running even when spikes do not occur, thereby wasting energy. If we have spare servers that are always active to handle spikes in load, we should be able to increase the load threshold used for signaling the start of a new server potentially saving energy.

Equation 8 shows the relationship between the load (L_t) at time t, the peak load in the interval D seconds from t ($\hat{L}_{t,t+D}$), the number of spare servers (S), and the threshold setting (T_S) with spare servers that needs to be honored to meet our QoS constraint. It is a variation of Equation 7 in the context of spare servers. It states that the total number servers at time t (including the spare servers) should be

Table 3: Spare Server Performance

	Financial							Olympics						
Spares	0	1	2	3	4	5	6	0	1	2	3	4	5	6
T(s)	100	125	155	194	258	388	776	89	119	179	241	358	482	965
Savings (%)	36.7	32.5	29.5	25.1	18.9	13.8	7.5	7.1	11.8	16.1	17.7	12.0	10.7	8.6
Predicted (%)	40.0	36.0	32.6	27.3	21.0	15.5	8.5	10.0	14.2	19.8	19.8	14.9	12.3	9.6
$S_{nostandby}$ (%)	39.7	36.1	32.7	27.1	21.0	15.5	8.65	10.1	14.3	20.9	19.9	14.9	12.4	9.6
S_{on} (%)	38.6	34.4	31.1	26.4	20.2	14.9	8.3	8.1	13.0	17.0	18.8	13.1	11.6	9.3

enough to keep the maximum load per server in the interval from t to $t + D$ from exceeding C. For a given load profile, we can see from Equation 8 that as S increases T_S can also increase.

$$\hat{L}_{t,t+D}/C \quad <= \quad S + L_t/T_S, for\ all\ time\ t \qquad (8)$$
$$N_{average} \quad = \quad (\sum_t (L_t/T_S + S))/Run\ Time \qquad (9)$$

Equation 9 shows the average number of servers when using spare servers. Based on the load profile, increasing the spare servers, *S*, can have a beneficial or detrimental effect on the energy consumption. A detrimental effect is possible because one may not be able to raise T_S enough (also dependent on the load) to compensate for the energy consumed by the spare servers. Using the constraint expressed in Equation 8, the load profile, the value for the startup delay *D* (30s), and the value for the capacity *C* (140 connections), we can compute $N_{average}$ for any value of *S* for both the workloads. Table 3 shows the T_S and energy savings, *Savings*, for both the workloads.

We see that Financial does not benefit from having spare servers at all, while Olympics does best with 3 spare servers. The *Predicted* row corresponds to the estimated energy savings computed as $\{1 - N_{average}/8\}$. While the actual energy savings is close to *Predicted* it is still a little less. A significant portion of this difference is due to some of the servers continuing to remain in the *standby* state because of active connections even after PARD determines they are not needed. The row S_{on} shows the savings 'estimated' from the actual number of servers on during the run. $S_{nostandby}$ shows the savings 'estimated' by counting the *standby* servers as off. The difference in the two numbers gives a measure of the error introduced is the prediction because of servers remaining in *standby* till activity ceases on their connections. The difference between S_{on} and *Savings* is largely due to the variation in energy consumption depending on the amount of activity on a server - the *on-off* model assumes a fixed energy consumption for a *busy* server irrespective of activity.

The benefits for Olympics from spare servers is because the load is high enough most of the time that the number of *required* servers is higher than the number of spare servers. Thus, the increase of $N_{average}$ due to S in equation 9 is negligible. For ST, the steep slope in the beginning of the Olympics profile requires a low threshold to ensure that average load does not exceed *C*. Using spare servers for this segment of the workload allows the threshold to be set much higher, resulting in significant savings during the rest of the load profile. Thus, spare servers can be found more beneficial than ST when the average load is high, and the load has occasional steep spikes that require it to have a low threshold in the absence of spare servers.

5.2.2 History-based Schemes

The idea of history-based schemes is that one could be familiar with the general characteristics of the load at a site and could potentially modify ST to incorporate this knowledge and come closer to achieving the maximum possible energy savings. In this section, we consider three levels of such information based on expected spikes in the workload. The first approach assumes just the knowledge of the maximum spike (S_{max}) encountered with the load. The second approach assumes the knowledge of the maximum spike encountered for each system configuration available: this corresponds to the number of servers required for each load point. The rationale behind having this knowledge is the capacity to handle spikes increases with the number of active servers in the cluster. At any point in the workload, the additional load that can be handled is $N * C - L$, where *L* is the load. The higher, the value of *N*, the greater the spike that can be tolerated. Hence, associating a value of *N* for each spike (S_N) in the load could help in better planning for starting additional servers. The third approach assumes we have perfect knowledge of the workload: the increase in load is known at every point in time. We will refer to the three approaches by the knowledge we have for each of them, as S_{max}, S_N, and S_t.

The relationship between the knowledge available and the prediction strategy for the number of servers required is shown in the equations 10 to 12. N_t is the number of powered-on servers (active plus those just turned-on) at time *t* and L_t is the load at time *t*. Using these equations, and the values for *D* (30s) and *C* (140 connections) for our platform, we can compute the number of servers required at any time *t* for each of the two workloads. Figures 10 and 11 show the number of active servers for each scheme.

$$N_t \quad = \quad (L_t + S_{max} * D)/C \qquad (10)$$

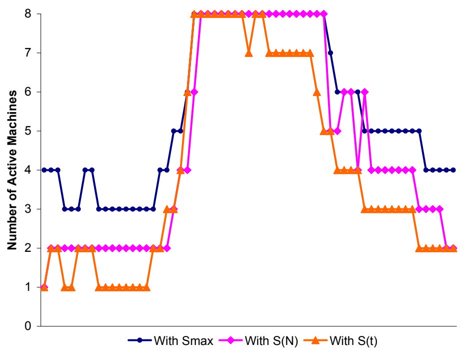

Figure 10: History-based schemes (Financial)

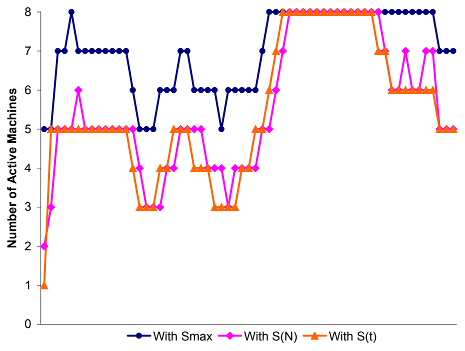

Figure 11: History-based schemes (Olympics)

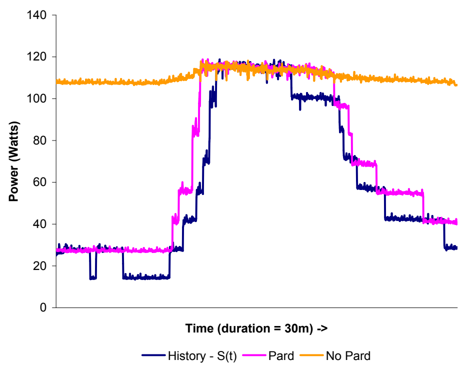

Figure 12: Energy using S(t) (Financial)

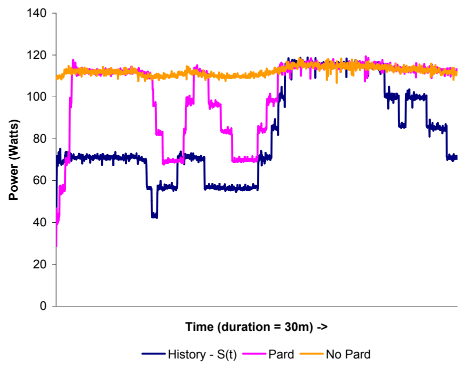

Figure 13: Energy using S(t) (Olympics)

$$N_t = (L_t + S_N * D)/C \tag{11}$$
$$N_t = (L_t + S_t * D)/C = \hat{L}_{t,t+D}/C \tag{12}$$

From these equations, we can compute the average number of servers active for the schemes S_{max}, S_N, and S_t, for Financial as 5.53, 4.68, and 4.07 respectively. They correspond to potential energy savings of 30.8%, 41.5%, and 49.2% respectively. Even the scheme with the best knowledge, S_t falls about 2% short of the maximum achievable savings (from the *required* curve for Financial in section 5.1) because any extra servers for the future need to be started D seconds (startup delay) earlier and consume that much extra energy. For Olympics, the potential energy savings for S_{max}, S_N, and S_t can be similarly computed to be 11.9%, 29.5%, and 30.4%, respectively.

Figures 12 and 13 show the actual energy consumption measured on our platform for S_t, the best history-based scheme, for the Financial and Olympics, respectively. They are given in contrast to the ST scheme and the no Pard scheme. The actual energy saved by S_t for Financial and Olympics compared to not using Pard are 45.6% and 26.1% which are quite close to the predicted savings of 49.2% and 30.4%, respectively. Exploiting additional knowledge of the system-workload context brings significant improvement over the original ST scheme which saved 36.7% and 7.1% for Financial and Olympics, respectively. The small discrepancies between the predicted and actual savings are largely due to the time a server spends in the *standby* state between the *busy* and *retire* states and due to the fixed active energy cost assumption for the estimates.

6 Conclusions

Power-aware request distribution attempts to provide significant energy savings in server clusters by matching resource utilization to the load imposed on the servers. The problem domain spans four dimensions - energy, quality of service, system characteristics and workload characteristics. We establish the interaction between the system-workload characteristics and energy savings, for the power-aware request distribution problem. We identify the key characteristics for the system and workload that influence the impact of energy-saving schemes. We show analytically the role of these characteristics and specific system-workload parameters on the maximum achievable energy savings. Our work shows that knowing the system-workload context is critical to understanding the value of any energy-saving proposals.

We verify our conclusions with actual energy measurements on our prototype for energy-conserving server clusters. To do so, we develop an effective method for generating workloads from a commercial benchmark (specifically using TPC-W) and real-world server logs. This addresses the need for two essential characteristics in workloads for this problem domain: (a) commercial/well-accepted workloads, and

(b) real-world variations in load profile.

We show that our model along with the proper understanding of the system-workload context can be effectively used to analyze not just our basic energy-conserving approach (Simple Threshold - ST) but other solutions too: (a) Spare Servers, which is a variation of the ST scheme, and (b) History-based schemes, which use the knowledge about the load profile to save additional energy. Our experimental studies verify our analytical conclusions for both the new classes of solutions.

In this study, our goal was to understand the role of the system-workload context on energy-saving schemes for server clusters. In future work, we would like to address workloads that are not connection-oriented - using the appropriate load metric in place of connections might suffice. We plan to study the effect of the system-workload context on the trade-offs between energy-savings and QoS. We are also interested in developing robust online algorithms for energy-savings which would utilize the important system-workload factors we identified in this paper.

Acknowledgements

We thank Alison Smith and Ravi Kokku of the University of Texas for their work on the PARD prototype implementation. We also thank Eric van Hensbergen, Freeman Rawson, Mike Kistler, Tom Keller and Ram Rajamony of the IBM Austin Research Lab for their help during this work.

References

[1] P. Bohrer, E. Elnozahy, T. Keller, M. Kistler, C. Lefurgy, C. McDowell, and R. Rajamony, "The case for power management in web servers," in *Power-Aware Computing* (R. Graybill and R. Melhem, eds.), Kluwer/Plenum Series in Computer Science, Jan. 2002.

[2] Hewlett-Packard Company, Inc., "Proliant BL e-Class System Setup and Installation Guide." http://www.hp.com/, December 2002.

[3] RLX Technologies, "RLX ServerBlade 1200i Data Sheet," December 2002.

[4] R. Credle, D. Brown, L. Davis, D. Robertson, and T. Ternau, "The Cutting Edge: IBM eServer BladeCenter." http://www.redbooks.ibm.com/, November 2002.

[5] J. Chase, D. Anderson, P. Thakar, A. Vahdat, and R. Doyle, "Managing energy and server resources in hosting centers," in *Proceedings of the 18th Symposium on Operating Systems Principles (SOSP)*, Oct. 2001.

[6] E. Elnozahy, M. Kistler, and R. Rajamony, "Energy-efficient server clusters," in *Proceedings of the Workshop on Power-Aware Computing Systems*, Feb. 2002.

[7] E. Pinheiro, R. Bianchini, E. V. Carrera, and T. Heath, "Load balancing and unbalancing for power and performance in cluster-based systems," in *Workshop on Compilers and Operating Systems for Low Power*, Sept. 2001.

[8] Transaction Processing Performance Council, "TPC-W: A Transactional Web E-Commerce Benchmark - http://www.tpc.org."

[9] M. Weiser, B. Welch, A. Demers, and S. Shenker, "Scheduling for reduced CPU energy," in *First Symposium on Operating Systems Design and Implementation*, (Monterey, California, U.S.), pp. 13–23, 1994.

[10] K. Flautner, S. Reinhardt, and T. Mudge, "Automatic performance setting for dynamic voltage scaling," in *Proceedings of the 7th ACM Int. Conf. on Mobile Computing and Networking (MOBICOM)*, July 2001.

[11] A. Buyuktosunoglu, S. Schuster, D. Brooks, P. Bose, P. Cook, and D. Albonesi, "An adaptive issue queue for reduced power at high performance," *Lecture Notes in Computer Science*, vol. 2008, May 2001.

[12] S. Kaxiras, Z. Hu, and M. Martonosi, "Cache decay: Exploiting generational behavior to reduce cache leakage power," in *The 28th International Symposium on Computer Architecture (ISCA-28)*, pp. 240–251, June 2001.

[13] A. Dhodapkar and J. E. Smith, "Managing multi-configuration hardware via dynamic working set analysis," in *Proceedings of the 29th International Symposium on Computer Architecture*, May 2002.

[14] A. R. Lebeck, X. Fan, H. Zeng, and C. S. Ellis, "Power aware page allocation," in *Architectural Support for Programming Languages and Operating Systems*, pp. 105–116, 2000.

[15] E. Elnozahy, M. Kistler, and R. Rajamony, "Energy Conservation Policies for Web Servers," March 2003.

[16] V. S. Pai, M. Aron, G. Banga, M. Svendsen, P. Druschel, W. Zwaenepoel, and E. Nahum, "Locality-aware request distribution in cluster-based network servers," in *Proceedings of the 8th ACM Conference on Architectural Support for Programmin Languages and Operating Systems (ASPLOS-8)*, Oct. 1998.

[17] K. Appleby, S. Fakhouri, L. Fong, G. Goldszmidt, M. Kalantar, S. Krishnakumar, D. Pazel, J. Pershing, and B. Rochwerger, "Oceano - SLA based management of a computing utility," in *Proceedings of the 7th IFIP/IEEE International Symposium on Integrated Network Management*, 2001.

[18] W. M. Felter, T. W. Keller, M. Kistler, C. Lefurgy, K. Rajamani, R. Rajamony, F. Rawson, B. Smith, and E. V. Hensbergen, "On the Performance and Use of Dense Servers," tech. rep., IBM, 2003.

[19] PCI Industrial Group, "CompactPCI Specifications, PICMG 2.16," Oct. 2001.

[20] W. Zhang, "Linux Virtual Server for Scalable Network Services," July 2000.

[21] C. Amza, A. Chanda, E. Cecchet, A. L. Cox, S. Elnikety, R. Gil, J. Marguerite, K. Rajamani, and W. Zwaenepoel, "Specification and Implementation of Dynamic Web Site Benchmarks," in *Fifth Annual IEEE International Workshop on Workload Characterization (WWC-5)*, November 2002.

[22] MySQL, "http://www.mysql.com/."

[23] Apache Software Foundation, "The Apache HTTP server." http://www.apache.org.

[24] PHP, "PHP: A HTML-embedded Scripting Language." Available at http://www.php.net, 2002.

[25] APC Community Connect, "APC: Alternate PHP cache." Available at http://apc.communityconnect.com.

[26] Red Hat, Inc., "TUX 2.2," 2001.