Multi-mode Energy Management for Multi-tier Server Clusters

Tibor Horvath
tibor@cs.virginia.edu

Kevin Skadron
skadron@cs.virginia.edu

Department of Computer Science
University of Virginia
Charlottesville, VA 22904

ABSTRACT

This paper presents an energy management policy for reconfigurable clusters running a multi-tier application, exploiting DVS together with multiple sleep states. We develop a theoretical analysis of the corresponding power optimization problem and design an algorithm around the solution. Moreover, we rigorously investigate selection of the optimal number of spare servers for each power state, a problem that has only been approached in an ad-hoc manner in current policies.

To validate our results and policies, we implement them on an actual multi-tier server cluster where nodes support all power management techniques considered. Experimental results using realistic dynamic workloads based on the TPC-W benchmark show that exploiting multiple sleep states results in significant additional cluster-wide energy savings up to 23% with little or no performance degradation.

Categories and Subject Descriptors

C.4 [**Performance of Systems**]: Performance attributes; D.4 [**Operating Systems**]: General

General Terms

Algorithms, Design, Experimentation, Management, Performance

Keywords

energy management, internet servers, multi-tier applications, reconfigurable clusters, end-to-end latency, dynamic power management, dynamic voltage scaling, sleep states

1. INTRODUCTION

Large data center operating costs arising from the steadily increasing energy consumption of recent server hardware have prompted the consideration of cluster-level energy management policies by researchers. Current solutions increase overall cluster energy efficiency through dynamic power management techniques such as DVS and dynamic cluster reconfiguration. DVS may increase the efficiency of individual machines by reducing their power and performance when they are not fully utilized. On the other hand, reconfiguration strategies reduce total cluster power by consolidating load on a subset of machines and turn off the remaining ones.

A key observation is that these existing policies are focused on energy management of the *active portion* of the cluster: based on the current load, they determine what the active capacity should be. The inactive (idle) portion is simply turned off (except for a few spare servers). However, newer systems already contain support for multiple sleep states with various latencies and standby power levels. Exploiting the benefits of these extra states requires energy management of the *inactive* portion, which allows minimizing the total energy consumption of sleeping machines subject to responsiveness constraints. The resulting *multi-mode* energy management solution can achieve the same level of responsiveness as existing policies using a single sleep state (Off) only, but with significantly less energy. Despite this potential, we are not aware of any research that has explored the benefits of multiple sleep modes. Yet our promising results indicate that multi-mode energy management might become an important necessity in future energy-aware clusters.

The main contribution of this paper is the presentation of a comprehensive theoretical analysis, design, and evaluation of multi-mode energy management for multi-tier server clusters. This includes developing the active energy optimization and the sleep energy optimization policies, as well as experimental studies on an actual cluster, in which we compare the energy efficiency and performance of several policies.

2. RELATED WORK

Pinheiro et al. [12] design a power management scheme for reconfigurable clusters based on distributing demand for resources to a subset of the cluster. Transient demand variations are compensated for with PID feedback control. They report energy savings of 45%, however their static web workload was network interface-bound with peak CPU utilizations of only about 25%. To provide some spare capacity to compensate for wakeup latency, they assume that machines have a lower than actual capacity. To control the amount of spare capacity, a heuristic parameter was used. Elnozahy et al. [4] propose cluster reconfiguration combined with DVS,

PACT'08, October 25–29, 2008, Toronto, Ontario, Canada.

assuming a cubic relation between CPU frequency and power, evaluated by simulation. They conclude that the cluster uses least energy when all machines operate in an optimal frequency range, not necessarily using the least number of active machines, which is in contrast with our observations (due to our different power model). The work by Rajamani and Lefurgy [14] is highly relevant to ours. They point out critical workload factors (e.g., peak load) that must be properly selected for valid evaluations of energy management schemes. Hence, we use a similar methodology in our evaluations. They also study how the availability of spare servers affect energy savings in different workloads, but do not attempt to derive the optimal number of spare servers.

Heterogeneous clusters were also addressed in the literature. Heath et al. [7] developed a solution based on a complex system model built from extensive characterization. Implementing an adaptive Web server that takes advantage of the model, they achieve 42% energy savings on a small cluster with two types of machines, serving mostly static requests. One shortcoming is that their model does not incorporate DVS. Rusu et al. [16] report 45% energy savings in a small heterogeneous cluster combining reconfiguration and local (independent) DVS. The rely on power, execution time, and server capacity characterization to provide energy management with soft real-time performance. They define the `max_load_increase` variable as the maximum slope of the cluster's load, based on which they precompute a table of load values at which additional machines must be turned on.

Complementary to our research are efforts by Ranganathan et al. [15] directed at dealing with cluster-level power provisioning. While our work addresses the power optimization problem under performance constraints, their work addresses the opposite: the performance optimization problem under power constraints. Fan et al. [5] employed power modeling to attack data center-level power provisioning inefficiencies. These approaches are useful for saving on power delivery and cooling infrastructure costs, while our work is useful for saving additional operational energy costs. Such solutions can be combined; Raghavendra et al. [13] investigate a coordinated design, which ensures that the various power and performance optimizing controllers at different levels of the data center hierarchy can correctly interoperate. However, their work does not discuss spare capacity or multi-mode energy management.

Our work is different from the above in that (*i*) we address multi-tier clusters with highly dynamic workloads, (*ii*) we perform active energy optimization using coordinated reconfiguration and DVS based on theoretical analysis, (*iii*) we use feedback control to correct for estimation errors, and (*iv*) our policy is evaluated with a realistic trace-driven benchmark with the peak load correctly sized to our cluster's capacity. Another key difference is that we address multiple sleep states. In general, it is not straightforward to extend the techniques presented in previous work to multiple states, because the new trade-off between wakeup latency and sleep power must be examined.

3. SYSTEM MODEL

We consider multi-tier server clusters, consisting of a constant number of tiers, each of which are composed of a variable number of machines. All machines in one tier run the same application, and requests go through all tiers. The end-to-end server performance must meet a predefined service level agreement (SLA). We assume a simple SLA specifying a target average end-to-end server delay D. Finally, each machine optionally supports multiple P-states (DVS).

In order to simplify the problem, we make a few important assumptions. First, we assume that machines within a single tier have identical power and performance characteristics; in other words tiers are homogeneous in terms of hardware. This is a reasonable assumption because, even though whole data centers are typically heterogeneous, it is normally preferred that groups of servers running the same application and being load balanced (i.e. the equivalent of a tier) are identical—this simplifies load balancing, among others. The second, related assumption is that there is perfect load balancing within a tier such that the CPU utilization of the constituent machines is equal. While this is idealistic, it is only required for analytical tractability, and based on our empirical observations the impact of moderate imbalances on actual performance is insignificant.

As an implication of the assumptions above, we do not need to model individual machine CPU frequencies—it is sufficient to use a single value per tier. This is because, since power is a convex function of frequency [8], setting the same frequency on all nodes within a tier is equally or more power-efficient than setting different values. Further, the performance effects of frequency scaling are not greater than its effects on power [9], therefore there is no performance or energy-efficiency benefit from different settings within a (homogeneous) tier either. Therefore, we limit our study to the model where each tier is operated at a single CPU frequency.

3.1 Power Model

In our system model, two variables have a significant effect on the power draw of an individual active machine: its CPU utilization and frequency (which also affects core voltage). Other variables may have an indirect effect by influencing these. To obtain a predictive model for machine power, P_i, we analyzed the actual power measurements from a large pool of characterization experiments, in which a machine operates at a varying CPU utilization (U_i) and frequency (f_i). Figure 1 shows the resulting curves. We found that P_i is approximately linear in both U_i and f_i for any fixed f_i and U_i, respectively. The relationship between power and CPU frequency is in reality slightly super-linear (as expected from the CMOS power model [11] assumed throughout the literature, e.g., [2, 4, 10]), however a linear approximation was sufficiently accurate for our purposes and preferable for its simplicity.

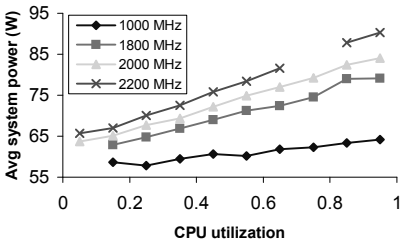
From analysis of the curves, we obtain our power model:

$$P_i(f_i, U_i) = a_{i3} f_i U_i + a_{i2} f_i + a_{i1} U_i + a_{i0}. \quad (1)$$

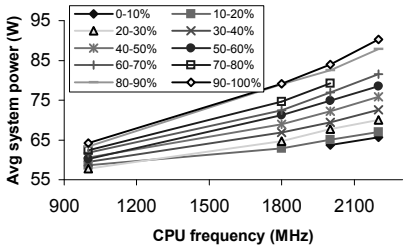
Parameters a_{ij} are determined by curve fitting. Power estimation for our test system using this model was fairly accurate (confirmed by the coefficient of determination $R^2 = 0.9879$), with average error of 1% (worst case $< 4\%$).

3.2 Service Latency Model

The service latency of short requests in concurrent servers is mostly a function of CPU utilization. Hence, in order to predict latency for various cluster configurations and performance states, we first need to model CPU utilization in those configurations. Given our assumptions on perfect load



(a) Average system power vs. CPU utilization.



(b) Average system power vs. CPU frequency.

Figure 1: Average system power measurements.

balancing and equal CPU frequencies in a tier, the relation between offered load λ_i, CPU utilization U_i, tier size m_i and frequency f_i can be estimated using:

$$\lambda_i = m_i f_i U_i. \tag{2}$$

Prediction is achieved through a two-step process: we first estimate the current λ_i from measurements, and then based on this we can predict U_i for the selected configurations (m_i, f_i). This model produced fairly accurate predictions in our test system ($R^2 = 0.9690$).

Building on this, service latency W_i can also be predicted for any tier configuration. To obtain our latency model, we performed nonlinear regression analysis using a heuristically decided format. After computing the fitting coefficients (via curve fitting to our measurements) we obtained the following model ($R^2 = 0.9968$):

$$W_i(U_i) = \frac{w_{i1}}{(1-U_i)^2} + w_{i0}. \tag{3}$$

4. MULTI-MODE ENERGY MANAGEMENT

In multi-mode energy management, the energy use of both active and idle nodes must be considered. In general, total cluster energy is given by:

$$E_{\text{total}} = E_{\text{active}} + E_{\text{sleep}} + E_{\text{transition}}. \tag{4}$$

Assuming a reasonable energy management policy, E_{active} is dictated by load and is not affected by the other terms. As long as sleeping machines can be waken up in time to meet demand, optimization of E_{sleep} can be treated as a separate concern. Minimization of $E_{\text{transition}}$ is considerably less important in typical Internet server workloads, since load fluctuations occur on a larger time scale (i.e., daily or longer). Prior works have ensured that it remains an insignificant fraction of total energy by smoothing out short bursts and imposing a minimum time between cluster reconfigurations [12, 7].

4.1 Active Energy Optimization

An energy management strategy is considered optimal if it assigns machines to tiers and determines their operating frequencies such that the total power consumption of the system is minimal while the SLA is still met. This is more formally expressed as the following minimization problem:

$$\min_{m_i, f_i} \quad P_{\text{tot}} = \sum_{i=1}^{s} m_i P_i \left(f_i, \frac{\lambda_i}{m_i f_i} \right) \tag{5a}$$

$$\text{subj. to} \quad W_{\text{tot}} = \sum_{i=1}^{s} W_i \left(\frac{\lambda_i}{m_i f_i} \right) \leq D \tag{5b}$$

$$\text{and} \quad \sum_{i=1}^{s} m_i \leq M. \tag{5c}$$

We solve the problem using the method of Lagrange multipliers. Setting $\forall i : \nabla_{m_i, f_i} L = 0$, where $L(m_i, f_i, l_1, l_2)$ is the Lagrangian function (with multipliers l_1 and l_2), we get for each i:

$$\frac{\partial L}{\partial m_i} = a_{i2} f_i + a_{i0} + l_1 \frac{\partial W_i}{\partial m_i} + l_2 = 0, \tag{6a}$$

$$\frac{\partial L}{\partial f_i} = a_{i2} m_i - a_{i1} \frac{\lambda_i}{f_i^2} + l_1 \frac{\partial W_i}{\partial f_i} = 0. \tag{6b}$$

Substituting Equation (3) and solving for l_1 results in our optimality criterion:

$$G(m_i, f_i) = (1-U_i)^3 \frac{m_i}{w_{i1}} \left(\frac{a_{i2} f_i}{U_i} - a_{i1} \right) = 2 l_1. \tag{7}$$

This means that, assuming the constraints are active (i.e., the total latency can potentially exceed the deadline), total cluster power use can only be optimal if G are equal across all tiers, more formally if:

$$\forall i, j : G(m_i, f_i) = G(m_j, f_j). \tag{8}$$

4.2 Sleep Energy Optimization

Machines support n system sleep states, each characterized by a distinct power level and wakeup latency. The power dissipation and wakeup latency of state i is denoted as p_i and ω_i, respectively. We assume that $\forall i, 0 < i \leq n : p_i \leq p_{i-1} \wedge \omega_i \geq \omega_{i-1}$, where state 0 is the `On` state with $\omega_0 = 0$. As an example, recent ACPI-compliant hardware may support up to 5 system sleep states (S-states).

With respect to the workload, we assume that load spikes can be unpredictable, but arbitrarily steep spikes need not be supported (note, accommodating a step increase in load would require *immediate* wakeup capability). Intuitively, a trade-off exists between system responsiveness and the ability to exploit sleep states, which should be up to the user to control. Hence, we define a system parameter called maximum accommodated load increase rate (MALIR, denoted as σ) to be the maximal rate of increase of the demand for cluster capacity that must be met despite any power management activity. For simplicity, we assume a CPU-bound workload with the demand and MALIR measured in CPU cycles per second (Hz). If another resource is the bottleneck, it can be defined for that resource analogously.

We minimize the E_{sleep} component through spare capacity optimization. Since it is decoupled from the other components, optimal steady-state energy is achieved by optimizing for power. This means, because of the property $p_i \leq p_{i-1}$, that each spare (unallocated) server should be put to the

deepest possible state (greatest i), subject to the MALIR constraint.

4.2.1 Feasible Wakeup Schedule

Our problem becomes that of determining the minimum number of spare servers for each sleep state such that if the offered load increases with rate σ, a feasible wakeup schedule still exists. Let $c(t)$ and $d(t)$ denote cluster capacity and demand, respectively, at time t. Further let t_0 be the time at which load begins to increase. Assume that both $c(t_0)$ and $d(t_0)$ are known (e.g., from measurements and estimation). Then, from the definition of σ:

$$d(t) = d(t_0) + \sigma(t - t_0). \tag{9}$$

In order to determine whether a feasible wakeup schedule exists, it is sufficient to consider the case when *all* spare servers are waken up at time t_0—if this does not result in a feasible schedule, then one does not exist. Let S_i stand for the number of spare servers in sleep state i, and $f_{\max}$ the maximum CPU frequency. For the case described, capacity can be obtained from the recursive formula:

$$c(t_0 + \omega_i) = c(t_0 + \omega_{i-1}) + S_i f_{\max}. \tag{10}$$

In other words, cluster capacity at the expiration of the wakeup latency of some state increases by the maximum capacity of all machines in that particular state. From this, and observing that $d(t)$ continuously increases while $c(t)$ is stepwise, a feasible wakeup schedule exists iff:

$$c(t_0 + \omega_i) \geq d(t_0 + \omega_{i+1}), \quad 0 \leq i < n. \tag{11}$$

4.2.2 Spare Servers

Since we want to put spare servers in the deepest possible sleep states, we examine the limit (equality) case of Equation (11) so that no slack time is allowed before waking up spare servers. Because this requires the set of greatest still feasible wakeup latencies, it also results in the smallest feasible overall power due to the monotonicity of ω_i and p_i in i. Substituting into Equation (10), we get:

$$d(t_0 + \omega_{i+1}) = d(t_0 + \omega_i) + S_i f_{\max}, \tag{12}$$

which, after simplification by applying Equation (9) and rearrangement, yields the optimal number of spare servers for each sleep state as follows:

$$S_i = \sigma \frac{\omega_{i+1} - \omega_i}{f_{\max}}. \tag{13}$$

The above formula is in the continuous domain, whereas *a*) cluster reconfiguration is typically performed periodically; and *b*) only whole machines can be transitioned to sleep states. Hence, the formula is discretized to obtain the final solution:

$$S_i^* = \left\lceil \sigma \frac{\mathcal{T}(\omega_{i+1}) - \mathcal{T}(\omega_i)}{f_{\max}} \right\rceil, \quad 0 \leq i < n, \tag{14}$$

where $\mathcal{T}(\omega) = \lceil \omega / T \rceil \cdot T$ and T is the control period. This accounts for both that wakeups will only be initiated, and that the cluster will only be reconfigured to include the awakened machines, at controller activation.

5. POLICY DESIGN

We designed energy management policies that take advantage of our predictive models and theoretical results. The policies are periodically invoked, when they obtain basic measurements from the system, apply the models, compute the optimal power states, and initiate the necessary power state transitions.

5.1 Active Capacity Policy

A simple approach to computing the optimal power states would be to perform an exhaustive search over the possible cluster configurations, selecting the one that best matches the optimality criterion derived in section 4.1. However, there are two issues with this. Firstly, the criterion is only a *necessary* condition of optimality, and may not be a sufficient condition. Secondly, an exhaustive search does not scale up to large cluster sizes, or to large numbers of possible frequencies. Therefore, to address these issues, we designed a greedy heuristic search algorithm to perform the optimization.

5.1.1 Heuristic Optimization

The algorithm relies on an additional assumption regarding the power characteristics of the hardware: it assumes that the static system power dissipation is large enough that turning on an additional machine and lowering the common CPU frequency such that CPU utilization is kept constant, never saves power. More formally:

$$(m_i + 1) P_i\left(\frac{\lambda_i}{(m_i + 1)U_i}, U_i\right) \geq m_i P_i\left(\frac{\lambda_i}{m_i U_i}, U_i\right), \tag{15}$$

which reduces to $a_{i1} U_i + a_{i0} \geq 0$. This is a very realistic assumption with current server hardware. For instance, the machines in our testbed clearly satisfy it, and prior work has relied on similar findings [12].

The optimization can be performed in two rounds. The first round finds the minimum number of machines in each tier so that the end-to-end performance constraint $W_{\text{tot}} \leq D$ is satisfied using the highest frequency for the machines. We start from computing a minimum allocation that only ensures that any one tier will not exceed the end-to-end deadline in itself. Then, we continue the search by incrementally assigning machines to tiers one by one, until the predicted total latency meets the constraint. At each step, we need to decide which tier to add a machine to. We base this decision on our optimality criterion: we choose the tier that, with the added machine, would result in the "most equal" values $G(m_i, f_i)$. More precisely, we choose the assignment for which $\text{StdDev}(G_i)$ is minimal. In the second round, tier frequencies are decreased one at a time in a similar fashion, as long as $W_{\text{tot}} \leq D$ is still met. The decision which tier to affect at each step, is guided by the same objective: select the change that minimizes $\text{StdDev}(G_i)$. (The two-round structure of our heuristic is similar to the queuing theory based heuristic proposed by Chen et al. [3].)

5.2 Spare Server Policies

We developed two policies for comparison: Optimal, based on our sleep energy optimization result (section 4.2), and Demotion, which gradually demotes (transitions into a deeper state) spare servers after some fixed timeouts. The fixed timeout-based predictive technique is the most common DPM policy [1], and it also addresses multiple sleep states, which are the reasons for our choice. To ensure a fair comparison, a wakeup scheduling algorithm is used together with both policies that always wakes up machines when needed to meet

the expected future load based on the MALIR parameter. The difference is that while the Optimal policy attempts to ensure that machines can be awakened by the time they are needed, the Demotion policy does not.

5.2.1 Optimal Policy

The design of the Optimal policy is relatively simple. An algorithm precomputes S_i^* for each system power state ($0 \leq i < n$) allowed by the user at initialization. Then, it is invoked in each control period with the list of nodes determined by the active capacity policy to be idle. The Optimal policy is responsible for assigning these to sleep states as necessary. We start by checking if there are more idle machines in the `On` state than the optimum S_0^*. If so, each surplus machine must be assigned to one of the sleep states. Going from shallow to deep states, we assign as many surplus machines to each as necessary for it to reach its optimum. Finally, any remaining surplus machines are assigned to state n.

5.2.2 Demotion Policy

The main idea of the Demotion policy is that whenever a machine becomes idle, a timer is started, and as it reaches each timeout the machine is demoted into the corresponding sleep state. There are two problems with directly applying this approach in a cluster. First, it is not scalable to maintain individual timers for each machine in a large cluster; and second, unless the activation of idle nodes is always done in a predetermined order (e.g., by rank), it is suboptimal. To see this, consider an example where demand fluctuates between 1 and 2 machines, but whenever it increases to 2, a different node is activated in a round robin fashion (e.g., for thermal load distribution). If this happens frequently enough, timers may never reach any timeouts, even though most of the cluster is always idle.

Therefore, instead of per-node timers, our solution is to just maintain the *count* of idle machines together with the time each smaller count was first seen. More precisely, we define a list of timestamps named `idle_since`, which is initialized empty and then maintained as follows. If the number of idle machines at the current time t is greater than the size of `idle_since`, then t is appended to its end as many times as the difference. If it is smaller, then as many elements as the difference are removed from the list's end. Then, for each list element e, its timeout for state i has been reached if $t > e + \omega_i$. Summing up the number of elements by the deepest state they reached, we obtain the timeout count for each state. Taking the difference between the number of nodes currently in each state and that state's timeout count, we get the number of surplus nodes (or deficit) in each state. Then, working our way from the `On` state to deeper states, surplus nodes are demoted to states with a deficit, filling up deepest states first.

5.3 Load Estimation

Estimation of the offered load is crucial for predicting latency in a target cluster configuration. However, calculating it from the current CPU utilization has a limitation: demand that is higher than total CPU capacity cannot be detected from utilization measurement because it saturates at 100% (and in practice even lower). Therefore, relying on this method only would result in unacceptable underestimation of offered load as the system got into saturation. Hence, other inputs are needed to detect this condition. We chose to add two types of performance monitoring: one to detect when response times exceed D, and another to detect service errors resulting from overload (e.g., timeouts), both meaning our estimate of λ_i is low. Once detected, we rely on feedback control to increase our load estimates, in order to quickly and reliably drive performance within the SLA specification. The addition of our feedback controller is also beneficial since it allows the performance target to be met despite errors in modeling, characterization inaccuracies, parameter variations, and practical issues such as server errors due to overload.

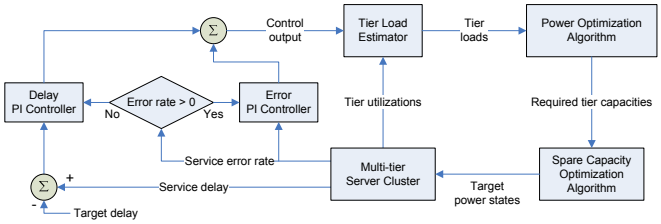


Figure 2: Design of the energy management policy with closed-loop control.

Our feedback control loop, sketched in Figure 2, is designed as follows. The controlled system (or "plant") is a multi-tier server cluster. Each period, three kinds of measurements are taken from it: tier CPU utilization, end-to-end service delay, and service error rate. These are all easy to obtain in practical systems: end-to-end response time samples are collected at the front-end tier and averaged across servers (weighted by the number of samples from each server), while error rates are summed. We employ two Proportional-Integral (PI) feedback controllers: one responds to latency violations, and one to service errors. (The reason to separate them is that they require different gains.) If service errors were observed, their rate is fed to the error controller, otherwise delay error is calculated by subtracting the target delay D from the measured average delay, which is fed to the delay controller. The final control output is produced by adding the selected controller's output and the other controller's integral component. Since the service error rate can never go below its target (i.e. zero), the integral component of the error controller is exponentially decayed (halved) when the rate is zero.

Next, the tier load estimator takes the utilization measurements and calculates an estimated offered load for each tier (section 3.2), adding the control output (if positive). The increment is distributed between reconfigurable tiers proportionally to their measured utilization, which ensures convergence to the true load plus any compensation added by the controller to correct steady state error. Then, the estimated tier loads are input to the active capacity policy (section 5.1), which determines the new energy-optimal allocation, reflecting minimal tier capacities required to maintain the required performance. Any extra machines are assigned to the appropriate power state by the spare server policy (section 5.2). Finally, the determined target power states are applied to the cluster, affecting the service and thereby closing the loop.

6. EXPERIMENTAL EVALUATION

6.1 Experimental Setup

6.1.1 Hardware

Our testbed uses a 12-node cluster composed of Linux-based PCs. Each PC has a DVS-capable AMD Athlon 64 processor, which supports the frequencies 1.0, 1.8, 2.0, and 2.2 GHz, as well as an Ethernet interface that supports the Wake-On-LAN protocol for remote wakeup capability. In addition, our PCs support four ACPI system power states: On (S0), Hibernate-to-RAM (S3), Hibernate-to-Disk (S4), and Soft-Off (S5). The machines also have 512 MB RAM, a hard disk, and a simple display adapter, but no monitor in order to keep the setup more realistic. A Watts Up Pro power meter is connected to the main power strip, measuring the AC power draw of the entire cluster.

6.1.2 Software

The cluster is set up as a 4-tier Web server consisting of the following tiers:

1. A front load balancer tier with 1 machine statically allocated. It runs the Linux Virtual Server (LVS) high-performance load balancer. The power draw of this tier was not measured because in a real-life cluster it would represent a much smaller fraction of total power than in our setup, thus our results would be skewed if we included it.
2. A Web (HTTP) server tier with dynamic machine allocation. This runs Apache with the `mod_jk` connector module, which forwards dynamic requests to the load balanced backend servers. We also developed a loadable module, which reports end-to-end latency samples to our controller.
3. An application server tier, also with dynamic machine allocation, which runs the JBoss enterprise Java application server. It uses the MySQL Java connector to forward queries to the backend database cluster.
4. A database cluster tier with 3 machines statically allocated. The MySQL Cluster scalable database package is used with 3 data nodes, over which the data set is partitioned. An SQL node (through which the cluster can be accessed) is co-located with each application server.

For each tier, standard versions of server software are used with no modifications necessary.

6.1.3 Policies

As our baseline, we consider a traditional cluster statically provisioned for peak load (i.e., not overprovisioned overall), which serves as our reference in energy efficiency and performance. This is first improved by applying our active capacity policy with only DVS enabled, modeling a non-reconfigurable cluster for comparison. Then, the two spare server policies are tested, each with both multiple states and with only the `Off` state allowed. This has two goals: to find out what additional gains can be realized by using multiple states over previous approaches that only exploit the `Off` state; and to compare the spare server policies themselves. The policies are listed in Table 1.

Table 1: List of policies compared.

Policy	Description
Baseline	Statically provisioned for peak load.
DVS	Baseline with DVS added.
Demotion (Off)	Demotion policy using the `Off` state only.
Demotion (Multiple)	Demotion policy using Multiple states.
Optimal (Off)	Optimal policy using the `Off` state only.
Optimal (Multiple)	Optimal policy using Multiple states.

6.2 Parameter Selection

6.2.1 Control Parameters

Our goal is to design a highly responsive controller, therefore we opt for an 8-second control period that can be considered very short in server applications. Hence, care must be taken when choosing the controller gains to avoid unstable behavior (i.e., oscillations between extremes). Since the controlled process is complex and dynamic, it is preferable not to rely on a detailed model when determining the gains. We use the classic Ziegler-Nichols tuning method, which although imperfect [6], has the advantage that it requires very little information about the system. In the final controller, the total integral component is clipped to between 0 and 8.8 GHz. Such clipping is typically performed to prevent excessive response in Integral controllers that may exhibit steady state error due to system constraints (e.g., not enough machines are available to reach the target). After experimenting with our system, we chose 250 ms for the average latency target, which is reasonable both in terms of user expectations of an interactive website, and compared to the 120–140 ms baseline with no energy management.

6.2.2 Optimal Timeout Values

We follow the methodology proposed by Benini et al. [1], who define the break-even time $T_{\mathrm{BE},i}$ for each sleep state i, which is the minimum inactivity time required to compensate for the transition energy. They suggest the optimal choice for timeout (yielding 2-competitive energy savings) is equal to $T_{\mathrm{BE},i}$. To calculate this for each state, characterization of the following parameters is necessary: time to enter and exit the state, average power while entering and exiting, and average power while in the state. From these, for the Demotion (Multiple) policy we obtain $T_{\mathrm{BE,S3}} = 20\,\mathrm{s}$ and $T_{\mathrm{BE,S4}} = 5980\,\mathrm{s}$. Here, the S5 state was found not beneficial because its power level is identical to that of S4 and thus it is excluded. Separate characterization is performed for the case when only the S5 state is used, yielding $T_{\mathrm{BE,S5}} = 114\,\mathrm{s}$ for the Demotion (Off) policy.

6.3 Trace-driven Dynamic Workload

A 3-tier implementation of the TPC-W benchmark was used as the test workload. It models a realistic bookstore website where all types of pages are dynamically generated using fresh data from the database, and also contain typical web images (buttons, logos, item thumbnails, etc.). The server establishes sessions with shopping carts, and ensures

Table 2: Description of Web traces used in our experiments.

Name	Length	Service
epa	1 day	The EPA WWW server at Research Triangle Park, NC; Aug. 30, 1995 (Wed).
sdsc	1 day	The SDSC WWW server at the San Diego Supercomputer Center, CA; Aug. 22, 1995 (Tue).
wcd37	1 day	1998 World Cup Web site; Jun. 1 (Mon).
wcw8	1 week	1998 World Cup Web site; May 3–9 (Sun-Sat).

all transactions within the same session are routed to the same application server (session stickyness). A separate client machine is used to generate load for the servers. It can emulate hundreds of independent emulated browsers (EBs), each navigating through pages as guided by a stochastic transition matrix. Retrieving a page and all images referenced by it is defined as one *web interaction*. Server throughput is measured in web interactions per second (WIPS), and latency in web interaction response time (WIRT).

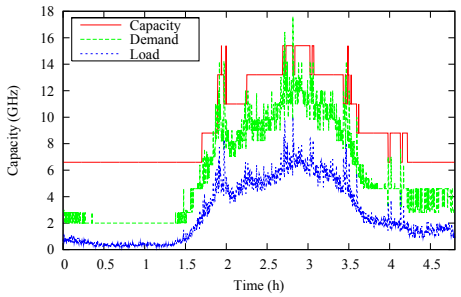
Instead of a static load as in performance-oriented studies, we are interested in performance and energy efficiency during realistic operation, which includes significant periods of off-peak load conditions. Hence, we simulate load variations found in real-life Web server traces, but using our dynamic benchmark—to create a trace-driven dynamic workload. We calculate the hourly average loads (in requests/s) for the trace in question, scale them by a constant factor to obtain the corresponding number of EBs such that the maximum equals the desired peak load, and then run experiments in which the number of EBs is varied over time to match the resulting sequence. Between data points, we linearly interpolate to achieve smooth fluctuations. A similar trace-driven approach was used by Rajamani and Lefurgy [14].

6.4 Results

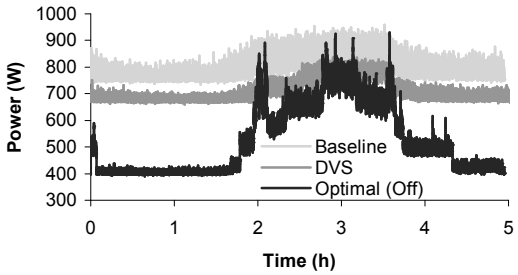
We perform comparison studies along three dimensions, to understand how certain aspects affect the performance of each policy. We consider different load profiles, then varying the peak load intensity, and the rate of load fluctuations. We experiment with several different traces listed in Table 2. The performance of each policy, including energy savings, energy-delay product (i.e. energy efficiency), and average end-to-end latency, is compared.

6.4.1 *Performance and Energy Efficiency*

We begin by validating the effectiveness of the active capacity management policy compared to the baseline setup and the DVS-only policy. The Optimal (Off) spare server policy is used so that results are comparable with previous ad-hoc spare server approaches. We ran the `epa` trace with peak load of 400 EBs (full capacity) and speedup factor of 5×, by which the day-long trace is simulated in 4.8 hours. Performance results are shown in Figure 3. As expected, our algorithm leaves sufficient margin over the estimated load when determining demand (Figure 3(a)), so that tier utilizations stay low enough to meet the target latency. Importantly, capacity follows demand without lag, which indicates the success of the spare server policy. Average latency for both the DVS and the Optimal (Off) policy was 227 ms, within the 250 ms target. (Baseline was 139 ms.) The power traces are compared in Figure 3(b). Looking at pure cost reduction, Optimal (Off) achieved 34% total cluster energy savings by dynamically turning off some machines when fewer could handle the load, compared to 11% with DVS only. To compare energy efficiency, we must factor in performance degradation as well. We use the energy-delay product, a widely used metric, defined as energy per request multiplied by the inverse of throughput (calculated from average power divided by the square of throughput). From this, we obtained 0.996 Js for Optimal (Off), a substantial improvement over the energy efficiency of both Baseline with 1.475 Js and DVS with 1.342 Js.



(a) Evolution of total cluster capacity (available machines), demand (optimal capacity as computed by the policy), and estimated load.



(b) Comparison of cluster power traces.

Figure 3: Power and performance results with the active capacity policy.

6.4.2 *Shape of Load Fluctuations*

Each trace has a unique shape of daily load fluctuations, giving rise to different energy saving opportunities. The peak load for each trace is identical (400 EBs), sized to total cluster capacity. All traces are accelerated 20× to shorten experiments. Results are shown in Figure 4. (Note, the data point of Demotion (Multiple) with `wcw8` is not available due to testbed instability.)

A number of key observations can be made. First, exploiting multiple states yields significant extra energy savings in the range 6–14%. Average gain is 10% for Demotion and 7% for Optimal. Second, the workload sensitivity of

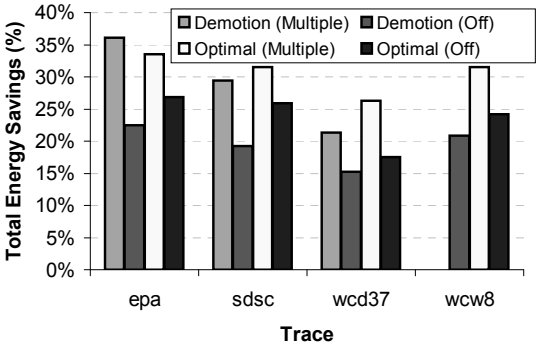


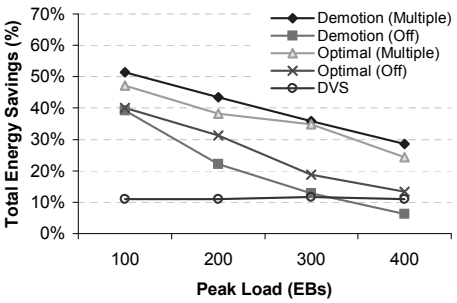
Figure 4: Cluster-wide energy savings of the experimental policies run against several traces, relative to the baseline.

Optimal (i.e., range of extra savings across workloads) is substantially smaller than that of Demotion (7–9% vs. 6–14%), which means a more predictable performance can be expected from it. Third, Optimal overall outperforms Demotion, by up to 7% in energy savings. The explanation of the one data point exception is that an extended period of `epa` has such a light load that no spare capacity is needed at all, giving an advantage to Demotion. The other traces have a slightly smaller dynamic range (in load level), where that advantage is lost, and Optimal wins by putting most nodes into deep sleep immediately instead of waiting for a long timeout. Finally, due to the common wakeup scheduling algorithm, all policies managed to maintain the expected performance with the fluctuations present in these workloads.

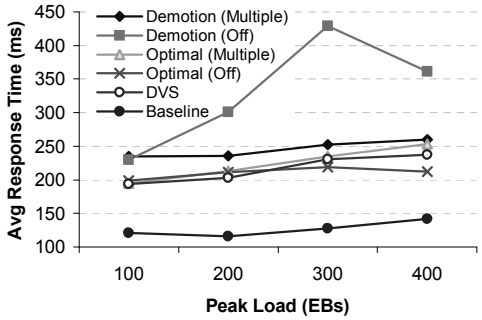
6.4.3 Peak Load Intensity

In many instances, the daily peak load does not approach cluster capacity, thereby increasing energy management opportunities. For example, there are weekly and seasonal load variations, or the cluster may be heavily overprovisioned. Although we cannot simulate realistic seasonal changes in reasonable time frames, we can run the same daily trace scaled to various peak levels to gain insight into policy performance at lighter loads. We use the `epa` trace accelerated to 60× and vary peak load from 100 (very light) to 400 EBs (full capacity). Results are shown in Figure 5.

Again, we observe significant extra energy savings from exploiting multiple states, in the range 7–23%; average gain is 20% for Demotion and 10% for Optimal. As seen in the previous results, for this trace Demotion (Multiple) outperforms Optimal (Multiple) by 1–5%. However, Demotion (Off) is highly inefficient—Optimal (Off) outperforms it by up to 9%. Moreover, with anything higher than very light load, it has *worse* energy efficiency than the baseline (1.81 J/s vs. 1.49 J/s at 400 EBs), and it also fails to meet the target latency (Figure 5(b)). This is caused by the policy's inability to meet demand in time, because it does not leave the necessary spare capacity to absorb load while machines are being waken up. In contrast, Optimal (Off) works with the same wakeup latencies, but has excellent performance because of its correct provisioning of spare servers. Note that there was no performance problem with Demotion (Off) in the previous experiments using a 20× speedup factor with the same trace. This raises the question of how this factor affects results, which we will explore in the next section. Also



(a) Cluster-wide energy savings compared to the baseline.



(b) Average client-perceived latency (WIRT).

Figure 5: Effects of varying the peak load intensity (from very light to full capacity) on energy efficiency and performance of different policies.

note that Demotion (Multiple) still has good performance. The reason is that, even though it does not ensure sufficient spare capacity, wakeup from S3 in our system is sufficiently fast (< 15 s) in practice to accommodate fluctuations even at 60× speed. Finally, it is worth noting that with peak load 1/4 of the full capacity, over 50% total cluster energy savings are achieved.

6.4.4 Time Scale of Fluctuations

Previous trace-based experimental studies on cluster energy efficiency have picked arbitrary acceleration factors in order to shorten experiments, without evaluating its impact on the results. However, different factors yield workloads where the time scale of fluctuations is different relative to system invariants such as wakeup latencies, service startup times etc., which cannot be accelerated. As we have seen in the previous section, this can lead to marked differences in policy performance. Therefore, it is imperative to study sensitivity with respect to the speedup factor. The `epa` trace is used again, peak load scaled to full capacity (400 EBs). Each policy is tested with several acceleration factors and the results are presented in Figure 6.

We observe that the speedup factor has a major influence on energy savings. The reason is that the higher this is, the greater the MALIR must be to maintain performance, which in turn requires more spare capacity, resulting in diminished energy savings. How optimally the policy handles that MALIR requirement, is what makes a difference in energy use. Conversely, with greater time scale (slower) fluctuations, little spare capacity is necessary, and the difference

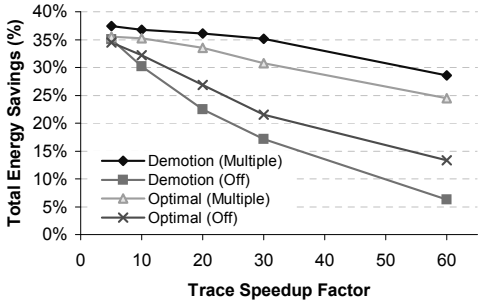


Figure 6: Effects of different trace acceleration factors on cluster-wide energy savings of different policies.

between policies becomes minor (assuming a MALIR-based wakeup scheduling algorithm is used).

7. CONCLUSIONS

Energy management for reconfigurable clusters can be separated into two independent concerns: the management of active and spare capacity. Spare capacity optimization minimizes energy of idle nodes subject to responsiveness constraints. In contrast with existing ad-hoc approaches only addressing a single `Off` state, we theoretically analyzed the problem in the context of multiple sleep states, and designed an optimization policy based on the results. Our results show that spare capacity optimization is especially important in Off-only systems with high rates of load fluctuation, where a Demotion-like naive policy should clearly be avoided because of significant performance degradation.

Our policies were validated by measurements on a realistic testbed with appropriate workloads and using actual hardware sleep states. The key result is that exploiting multiple sleep states affords substantial energy benefits. Extra energy savings of up to 23% over the Off-only version of the same policy were observed in highly dynamic workloads with negligible performance impact. Further, total energy savings over 50% were measured with lighter load. In addition, we found that the Optimal policy is superior to Demotion because it: (*i*) outperforms Demotion in energy efficiency with the majority of traces; (*ii*) avoids unavailable periods inevitable with Demotion due to late wakeups; (*iii*) guarantees responsiveness to the user-specified level (MALIR) with the least amount of energy; and (*iv*) does not rely on workload prediction, and therefore handles unexpected bursts and typical expected load fluctuations equally well.

An important implication of our results is that energy savings reported from policy studies using different trace acceleration factors may not be directly comparable (unless equal spare capacity was provisioned). A sensitivity analysis with respect to this factor is desirable to help interpret such results. Further, it is not clear how results from accelerated traces in small testbeds apply to real-time workloads in large-scale clusters because, even if spare capacity is the same, it represents a very different fraction of total cluster energy.

8. ACKNOWLEDGMENTS

This research was supported in part by NSF grant nos. CNS-0306404 and CNS-0615277, Army Research Office grant no. W911NF-04-1-0288, and a grant from Intel MRL. We would like to thank Tarek Abdelzaher for helpful discussions in the development of our approach and the anonymous reviewers for their constructive feedback.

9. REFERENCES

[1] L. Benini, A. Bogliolo, and G. D. Micheli. A survey of design techniques for system-level dynamic power management. *IEEE Trans. VLSI Syst.*, 8(3):299–316, 2000.

[2] P. Bohrer, E. N. Elnozahy, T. Keller, M. Kistler, C. Lefurgy, and R. Rajamony. The case for power management in web servers. In R. Graybill and R. Melhem, editors, *Power-Aware Computing*, Kluwer/Plenum series in Computer Science. Kluwer Academic Publishers, Jan. 2002.

[3] Y. Chen, A. Das, W. Qin, A. Sivasubramaniam, Q. Wang, and N. Gautam. Managing server energy and operational costs in hosting centers. *SIGMETRICS Perform. Eval. Rev.*, 33(1):303–314, 2005.

[4] E. Elnozahy, M. Kistler, and R. Rajamony. Energy-efficient server clusters. In *Proc. Workshop on Power-Aware Computing Systems*, Feb. 2002.

[5] X. Fan, W.-D. Weber, and L. A. Barroso. Power provisioning for a warehouse-sized computer. In *Proc. 34th Annual ACM/IEEE International Symposium on Computer Architecture*, pages 13–23, 2007.

[6] C. Hang, K. Astrom, and W. Ho. Refinements of the Ziegler-Nichols tuning formula. *IEE Proceedings D, Control Theory and Applications*, 138(2):111–118, Mar. 1991.

[7] T. Heath, B. Diniz, E. V. Carrera, W. M. Jr., and R. Bianchini. Energy conservation in heterogeneous server clusters. In *Proc. 10th ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming*, pages 186–195, 2005.

[8] C. hsing Hsu and W. chun Feng. When discreteness meets continuity: Energy-optimal dvs scheduling revisited. Technical Report LA-UR 05-3104, Los Alamos National Laboratory, Feb. 2005.

[9] C. Lefurgy, K. Rajamani, F. Rawson, W. Felter, M. Kistler, and T. W. Keller. Energy management for commercial servers. *IEEE Computer*, 36(12):39–48, 2003.

[10] A. Miyoshi, C. Lefurgy, E. V. Hensbergen, R. Rajamony, and R. Rajkumar. Critical power slope: understanding the runtime effects of frequency scaling. In *Proc. 16th International Conference on Supercomputing*, pages 35–44, 2002.

[11] T. Mudge. Power: A first-class architectural design constraint. *IEEE Computer*, 34(4):52–58, 2001.

[12] E. Pinheiro, R. Bianchini, E. Carrera, and T. Heath. Dynamic cluster reconfiguration for power and performance. In L. Benini, M. Kandemir, and J. Ramanujam, editors, *Compilers and Operating Systems for Low Power*. Kluwer Academic Publishers, 2002.

[13] R. Raghavendra, P. Ranganathan, V. Talwar, Z. Wang, and X. Zhu. No "power" struggles: coordinated multi-level power management for the

data center. *SIGARCH Comput. Archit. News*, 36(1):48–59, 2008.

[14] K. Rajamani and C. Lefurgy. On evaluating request-distribution schemes for saving energy in server clusters. In *Proc. IEEE International Symposium on Performance Analysis of Systems and Software*, pages 111–122, 2003.

[15] P. Ranganathan, P. Leech, D. Irwin, and J. Chase. Ensemble-level power management for dense blade servers. *Proc. 33rd Annual ACM/IEEE International Symposium on Computer Architecture*, pages 66–77, 2006.

[16] C. Rusu, A. Ferreira, C. Scordino, and A. Watson. Energy-efficient real-time heterogeneous server clusters. In *Proc. 12th IEEE Real-Time and Embedded Technology and Applications Symposium*, pages 418–428, 2006.