RC24791 (W0904-112) April 27, 2009
Computer Science

IBM Research Report

PADD: Power-Aware Domain Distribution

Min Yeol Lim[1], Freeman Rawson[2], Tyler Bletsch[1], Vincent W. Freeh[1]

[1]North Carolina State University
Raleigh, NC 27695-8206
USA

[2]IBM Research Division
Austin Research Laboratory
11501 Burnet Road
Austin, TX 78758
USA

Research Division
Almaden - Austin - Beijing - Cambridge - Haifa - India - T. J. Watson - Tokyo - Zurich

PADD: Power-Aware Domain Distribution*

Min Yeol Lim† Freeman Rawson‡ Tyler Bletsch† Vincent W. Freeh†
† North Carolina State University
North Carolina State Univ.
Raleigh, NC 27695-8206
{mlim,tkbletsc,vwfreeh}@ncsu.edu

‡ IBM Austin Research Laboratory
11501 Burnet Road
Austin, TX 78758
frawson@us.ibm.com

Abstract

Modern data centers usually have computing resources sized to handle expected peak demand, but average demand is generally much lower than peak. This means that the systems in the data center usually operate at very low utilization rates. Past techniques have exploited this fact to achieve significant power savings, but they generally focus on centrally managed, throughput-oriented systems that process a single fine-grained request stream. We propose a more general solution — a technique to save power by dynamically migrating virtual machines and packing them onto fewer physical machines when possible. We call our scheme Power-Aware Domain Distribution (PADD). In this paper, we report on simulation results for PADD and demonstrate that the power and performance changes from using PADD are primarily dependent on how much buffering or reserve capacity it maintains. Our adaptive buffering scheme achieves energy savings within 7% of the idealized system that has no performance penalty. Our results also show that we can achieve an energy savings up to 70% with fewer than 1% of the requests violating their service level agreements.

1 Introduction

Modern data centers usually have computing resources sized to handle expected peak demand. However, peak demand events are rare, and average demand is generally far less than peak. Although there is no systematic analysis of server utilization across wide range of data centers, a recent study estimates the average utilization to be between 5% and 20% [28]. Even in highly customized IT environments, it is hard to achieve 50% utilization. Therefore, a large number of computing resources are idle much of the time. This presents an opportunity for power savings if we can shut down or hibernate physical machines not needed for the current workload while still providing the required continuity of service. Fortunately, several modern virtualization systems [4, 5] provide the ability to migrate running virtual machines without interruption. Combining live virtual machine migration with the ability to power physical machines on and off turns the data center into a dynamically resizable pool of compute resources.

Powering servers on and off is a well-studied technique for *fluid* workloads such as transaction processing. Such workloads consist of many, short-lived, independent requests which are serviced by homogeneous software running on multiple physical systems. However, many environments do not run fluid workloads.

*This research was funded in part by an University Partnership award from IBM.

Instead, they execute coarse-grain workloads, such as database, batch, hosting or business logic systems which do not use a single workload distributor or which require heterogeneous software. Increasingly, the performance requirements of such workloads are specified in terms of *service level agreements (SLAs)* which give the throughput requirements for each application in the computing environment.

In this paper, we use simulation to evaluate the power and performance effects of dynamic workload consolidation for such workloads in a virtualized environment. The workloads are separated into multiple *domains*.[1] Data center power is reduced by packing multiple domains onto fewer physical nodes during periods of low utilization and expanding them back onto more systems when the usage rises. We call this coarse-grained packing scheme *Power-Aware Domain Distribution (PADD)*. In this paper, we explore the energy savings and performance impact of running heterogeneous domains in terms of demand and SLA.

One relatively new environment that may benefit from the PADD approach is the *Utility Grids*, which are clusters that are time- and space-shared by clients who will run an arbitrary workload. Amazon's Elastic Computing Cloud (EC2) [1], NCSU's Virtual Computing Lab (VCL) [3], and IBM Blue Cloud [2] are examples of this kind of grid. A key feature is that unlike HPC or business grids, the workloads that will be executed are not necessarily run-to-completion or throughput-oriented, but rather a mix. Some domains may run HPC-style apps and basically have a demand of 100% for the entire runtime. Others may be throughput-oriented and have fluctuating demand. Finally, some may act as human interfaces (shell, VNC, etc.), and have mostly low levels of demand. Given that the heavy use of virtualization by such grids, we expect to see a power-savings benefit from using PADD's domain packing.

The simulation tool used in this paper simulates dynamic domain packing based on traces of real-world workloads. It predicts both the power saved and the performance impact of a using PADD with a particular set of parameters for a particular workload as represented by a trace. By using a variety of traces, we are able to assess how generally applicable PADD is. The simulation also allows us to study how sensitive the results are to particular parameters such as buffer sizes and the time that it takes to migrate a partition or turn off a machine.

Our key contributions in this research are:

1. We developed a dynamic algorithm for placing domains to minimize total energy while avoiding SLA violations. Its key feature is the use of a two-level, adaptive buffering scheme which reserves capacity. Our scheme absorbs workload surges to avoid SLA violations, but adapting the buffer sizes to the workload reduces the amount of energy wasted on the buffers.

2. We provide a trace-driven simulation study of the energy benefits and the performance effects of our algorithm along with studies of its sensitivity to various parameters.

The rest of this paper is organized as follows. In Section 2, we discuss related work. Section 3 presents our model of a computing environment with coarse-grain workloads, and Section 4 gives a precise description of the PADD algorithm, discusses the parameters affecting it and presents the basics of our simulation. Section 5 presents our simulation results while Section 6 suggests future work and concludes.

2 Related work

Much research has been conducted on dynamic mapping of domains to physical machines. In some work, performance is the primary concern without power analysis. The work by Steinder, *et al*. [25], is both implementation and a simulation. In her work, there is a workload flow controller that manages the

[1] Xen [5] uses the term *domain*, which is used in this paper as a general, conceptual term for ease of exposition. Other terms in common use are *virtual machine (VM)* and *partition*.

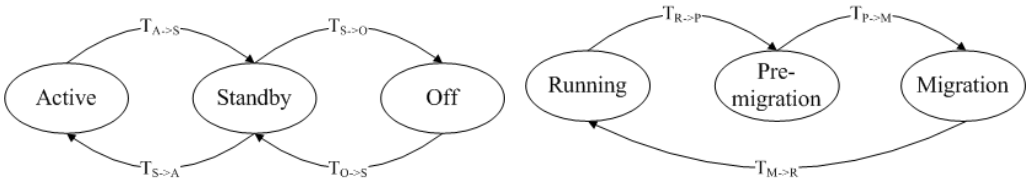
workloads being run and helps drive the application placement controller. In our work, there is no such workload controller: the workloads are run in the individual domains without any centralized control over them. The study by Bobroff, *et al.* [7], is a predictive placement scheme based on time series analysis. In contrast, our work is reactive, observing load and reacting appropriately.

The problem of saving power in a throughput-computing environment has also been studied previously. Most work centers on fluid workloads, i.e., workloads consisting of many short-lived independent requests that can be served by homogeneous software running on multiple physical machines and which have a centralized workload distributor that routes the requests to the machines which process them. Elnozahy, *et al.* [13] simulate the effects of dynamic voltage scaling coupled with vary-on/vary-off ("VOVO", basically powering nodes on/off). They run their simulator against two well-known production web traces and find that VOVO alone saves up to 42 % while the addition of voltage scaling saves an additional 18 %. Similarly, Chase, *et al.* [9] use an artificial resource economy to save energy by balancing the cost of resources against the benefit realized by employing them. This is also simulation-based work and focuses on satisfying the demand presented by web traffic traces. Both techniques save power by turning some nodes off and using a central workload distributor to route new requests to other nodes. The Power-Aware Request Distribution (PARD) system addresses data center power by using the individual request as the fundamental unit of work [23]. The system directs requests to a pool of compute resources which can be powered on or off depending on anticipated load. Because requests are independent and servers are stateless, powering down a server is simply a matter of removing it from the load balancer, waiting for pending requests to finish, and turning it off. In this environment, there is nothing to migrate or back up when shutting down a node, which leads to a very fine-grained power management: the system runs only the machines needed to satisfy the current demand. However, the PARD approach is only applicable to workloads similar to web-serving, and many workloads cannot be modeled in this manner. For example, processes may need to maintain state, or there may be multiple indivisible processes that each handle a separate incoming demand. In contrast to PARD, we study power management for a coarse-grained environment where the unit of work is a domain. We assume that domains have long lifetimes and contain state which must be maintained. Our approach is to migrate entire domains between nodes, instead of request distribution. This introduces two qualitative changes to the problem.

1. Unlike redirecting requests, there is a significant time and energy cost to migrating domains.

2. The number of domains does not change (or at least changes slowly) even though the computational demand of each domain may change quite rapidly.

Recently, Nathuji and Schwan [20] proposed a framework to determine the power-performance tradeoffs of virtual machines and to efficiently manage physical machines within given power budgets. This work focuses on achieving good performance under a power cap, whereas our work seeks to satisfy an SLA while minimizing energy consumption. Verma, *et al.* [26, 27] designed and evaluated several domain placement algorithms on top of their existing platforms. Like our approach, they examined the tradeoff between power consumption and migration cost of running domains when determining their placement in virtualized systems. Although the problem they are addressing is similar, our contribution is distinguished in two ways. First, because our work is a simulation study, we can define and explore numerous configurable parameters that vary by hardware and software characteristics and evaluate their effects in theoretical environments, including very large scale computing systems. Second, we propose a two-level buffering scheme where the buffering sizes are adapted by given SLA requirements. Our algorithm finds power efficient placement of domains while minimizing the SLA violation.



(a) Node states (b) Domain states

Figure 1. Node and Domain state diagrams. Each edge indicates the transition time delay.

There has been a significant amount of research into understanding and predicting workload patterns. Bradley, *et al.* [8] proposed two power management algorithms using short-term and long-term workload predictions. The algorithms identify temporal workload patterns and provide enough available resources proactively by exploiting these patterns. More recently, Chen, *et al.* [10] studied incoming traffic patterns and used them to predict the amount of resources needed to process the load. Such prior work focuses on improving predictions of workloads and applying the information to power aware server provisioning. In contrast, PADD simply reacts to the work being processed or being generated within the computing environment. In addition, these studies assume that one server is dedicated to a service. Since the servers are not shared by different services, there is no domain migration between running servers. These approaches are complementary to our work because combining prediction with reaction to the workload can lead to better placement decisions, achieving more power savings with less performance impact.

In contrast to coarse-grained domain packing, CPU packing and node packing techniques [6, 14, 16] are available to the operating system as a fine-grained packing. This allows the packing of jobs onto a subset of the available processors and power down the rest.

There is a large body of work on Dynamic Voltage and Frequency Scaling (DVFS) to save energy [15] [18]. The work [19] proposes a online power management approach on a virtual machines supporting isolation between them and honoring the power management requests of the individual virtual machines. Using DVFS allows us to improve further the power efficiency of the running systems in the environment by reducing voltage and frequency when a server's processors are not a bottleneck. Since our work aims at domain distribution in virtualized server environments, any local DVFS algorithm in a system can be combined with our approach to get additional power savings.

3 Model

This section offers some quantitative basis for our approach as well as defining the basic structure of the power and performance model used in our simulations.

3.1 Machine State and Virtualization

PADD distributes domains onto the available physical nodes using dynamic migration and controls the number of physical nodes by turning them on and off. Alternatively, if the systems support hibernation, entering and exiting hibernation state can be used to manage the number of active machines. Since our research aims at achieving power savings by packing multiple domains onto fewer physical nodes during periods of low demand, PADD tries to keep energy consumption as low as possible while meeting given performance requirements.

Running multiple domains with heterogeneous workloads may introduce additional overhead in the current virtualization technologies. The overhead mainly comes from sharing device accesses, expensive TLB misses, page faults, and memory handling [12]. Padala, *et al.* [22] have evaluated the overhead of the existing

Domain parameters	
M	Number of domains
$D_i(t)$	CPU demand of domain i at time *t*, $1 \leq i \leq M$
DS_i	Domain state in {*Running, Premigration, Migration*}.
$T_{DS_i \rightarrow DS_j}$	Domain state transition time
Node parameters	
N	Number of physical machines
C_i	Processing capacity of machine i, $1 \leq i \leq N$
NS_i	A machine state in {*Active, Standby, Off*}.
$P^i_{NS_i}(U)$	Power consumption of machine i at U utilization in the state NS_i, $1 \leq i \leq N$
$T_{NS_i \rightarrow NS_j}$	Machine state transition time
Eff_i	Maximum power-performance efficiency of machine i, $1 \leq i \leq N$
Performance parameter	
$SLA_i(d, l)$	d% of demands of domain i, must be processed within l seconds. $1 \leq i \leq M$

Table 1. Parameters defined for PADD and its power and performance model

virtualization technologies. They conclude that the overhead depends heavily on the running applications, their resource utilization, and the virtualization mechanisms used. Since virtualization is widely accepted in the industry, we make no attempt to model it in our simulation.

3.2 Model Description

We define the power and performance models used in our algorithm and its simulation. The diagrams in Figure 1 show the operating states and the transitions between them for nodes and domains respectively. We first define the operating states and the transitions between them for nodes and domains. For a node, there are three (3) stable states: *Active (A)*, *Standby (S)*, and *Off (O)*. Real systems may define more states, but we abstract them and collapse them for simplicity. In many modern systems, there is a variety of intermediate power states between *Active* and *Off*. These intermediate states consume varying amounts of power and typically trade off power consumption to reduce the time to get back to the *Active* state. However, for simplicity, we only consider one intermediate state, *Standby*.

Nodes may make four (4) transitions between states – *Active* to *Standby* (*A*→*S*), *Standby* to *Active* (*S*→*A*), *Off* to *Standby* (*O*→*S*), and *Standby* to *Off* (*S*→*O*). The transition time is a hardware-dependent characteristic of the particular systems being used in the installation. One key question is how strongly the power and performance behavior of PADD depends on the transition times between operating states.

Each domain can be in one of three (3) operating states: *Running (R)*, *Premigration (P)*, or *Migration (M)*. We assume that domains are long-lived entities and do not consider transitions to an off state. A domain is in the *Premigration* state during the time that the system is preparing to move it to a different node by copying data from the current node to the new one. Clarke [11] measured the overhead imposed by this state on a Xen platform running SPECWeb and found little measurable degradation, so our model assumes that performance in unaffected by premigration. However, a domain in the *Migration* state generally becomes out of service, and the downtime varies depending on workload. We assume that Migration stops the work on the source node and restarts it on the target resulting in a pause in processing. Our model assumes that *Migration* stops the work in the current node and restarts it in the target node, and it is accompanied by pause which causes a processing delay for current requests. The Xen measurements indicates that this delay is less

than 1 second. Consequently, frequent migrations will result in significant performance degradation. We take this gap in processing into account in our simulation. We use our simulator to analyze the sensitivity of PADD to different migration times.

The capacity of a node is the processing done by a single CPU running at maximum frequency. The CPU frequency is fixed – we do not model DVFS and related techniques, which are largely orthogonal to this work. The power consumption is taken to be a function of the system utilization. Since DVFS can affect both the capacity and power consumption, we assume for simplicity that the operating frequency is fixed at nominal. Also, most environments are heterogeneous with systems with processors of varying capacities. In [24], the capacity is modeled as a multidimensional vector of the available components (CPU, Memory, IO, *etc*). For simplicity, we assume that processing capacity can be normalized to a single value representing the CPU capacity. We hope to incorporate the insight of [24] in the future.

The performance requirement of a domain is specified by the service level agreement which is defined as $SLA(D, L)$. It specifies how much of the demand D (expressed as a percentage of the total) must be processed within a given time limit L (specified in seconds). Each domain may have its own SLA. At present, PADD constitutes a best effort to meet the given SLA, so violations are possible. In future work, we plan to bound the degree of SLA violation so that hard limits can be enforced.

Our problem is then to reduce energy consumption while offering the performance required by the SLAs. Finding the optimal domain distribution on every cycle is an example of a bin-packing problem, where the bin is a node and the bin's size is the node's capacity. This is an NP-hard problem [21]. The task is actually more nuanced than the abstract bin packing problem, because node transitions and domain migrations have time delays. Moreover, the number of active nodes is not fixed.

PADD is a heuristic method which approximates the optimal solution to the problem. Its key feature is that it uses two separate safety margins – per-node and whole system. We refer to these safety margins as *buffers*. The buffers are reserve capacity which allows PADD to deal with changing workloads and the latencies of the actions that it takes. Although the algorithm has a number of other parameters, the buffer sizes have the greatest impact. It turns out that the behavior of PADD is most sensitive to the amount of local buffering within each machine and the size of a global buffer of reserve capacity in the environment: the other parameters matter less. Our simulation offers a quantitative insight into our scheme and the parameters that we use to tune it.

4 Implementation

We define a number of policies affecting power and performance. Our simulation framework can parameterize these policies as a group of parameter settings. The policies can be categorized into three groups: buffering, time cost, and other decision policies.

4.1 Buffering

The PADD framework uses a two-level buffering scheme to deal efficiently with variations in the workload.

1. Local Buffer (LB): A reserved portion of the processing capacity on a single node. Local buffer sizes are specified in fractions of a CPU, so that reserving 10% processor's worth of capacity gives a local buffer size of 0.4 if the capacity (C) is 4.0.

2. Global Buffer (GB): An separate reserve pool of processing capacity spread across the active machines in the environment. Global buffer sizes are also specified in fractions of a CPU. If a quad-core

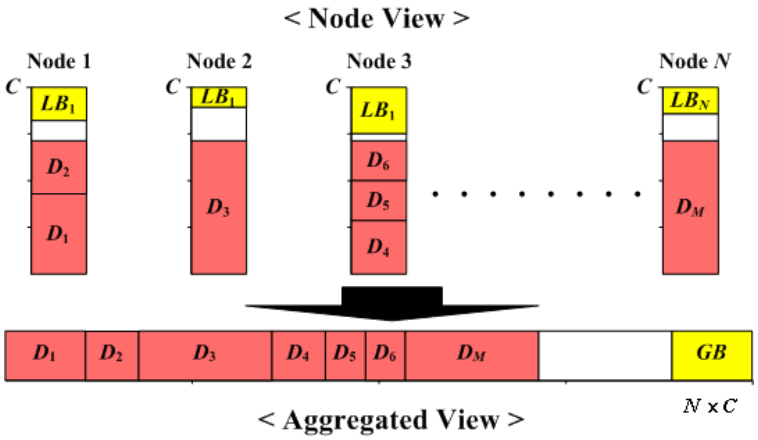


Figure 2. Each node has its own Local Buffer. The Global Buffer is a pool of additional capacity spread across the active nodes.

machine is modeled and each core has 1.0 capacity, reserving a whole machine yields a global buffer size of 4.0.

The Figure 2 illustrates how the buffering mechanism works. The local buffer prevents frequent and unnecessary domain migrations by absorbing smaller, transient variations in the aggregate demand of the domains on the machine. If there are k domains running in the node (x), when the aggregate CPU demand of all the domains starts to use LB, that is $C_x - \sum_{i=1}^{k} D_i(t) \leq LB$, it triggers an event to spread the demand over the physical nodes by migrating domains. However, in the interim, before any migrations can occur, the local buffer absorbs some or all of the additional demand. Using local buffers protects the SLA in the event that the aggregate demand is changing rapidly. In addition, the global buffer makes additional capacity ready to deal with larger, more permanent changes caused by domains starting new workloads or going to new workload phases. Therefore, the migration delay caused by node transition can be avoided. Both buffers handle surges, but they handle different types of surges.

In addition to supporting fixed buffer sizes, our simulator also supports an adaptive buffering scheme. In order to control the buffer sizes dynamically, the system monitors any SLA violations in a node and across the whole environment. We then calculate the GB and LB for each node using the following equations. If an SLA violation occurs on a node, we increase the LB size of the node. Similarly, we increase the GB size when there is any SLA for any domain across all of the nodes. The GB and LB are dynamically determined as a function of SLA violations.

1. $GB = \alpha \times Violation(all\_domains)$
2. $LB_i = \beta \times Violation(active\_domains_i)$

The number of violations among domains on a node in the active state is counted when determining the local buffer to avoid frequent changes in the LB size due to domain migrations. Increasing α and β gives less energy savings but allows for stricter adherence to the performance requirements. In our simulations, α and β are empirically chosen as 1 and 0.1. These parameters are derived by finding the best non-adaptive test cases and determining the values that would guide the adaptive system to similar results. We note that this is not necessarily optimal and that it is worth further study. We evaluate the power and performance tradeoffs of the two types of buffering in the next section.

4.2 Time Cost of Transitions

As defined in Section 3, there exist four (4) transition states for a node. The transition time of each state is specified as input in our simulation. For a domain, *Premigration* and *Migration* states are considered to be temporary, transitional states. The time spent in these two states also can be set as configurable parameter. In reality, the time delay in these two states are a function of the intensity of workloads, along with virtualization technology.

Turning a node off always reduces the overall power consumption of the computing environment. During node transitions, the machine continues to consume power. We assume for modeling purposes that the power consumption is constant for the entire duration of the transition and that the node does no useful computing during a transition. Moreover, the power during transition is generally higher than the power consumed when the machine is active but idle. As a result, we expect that frequent node transitions will result in more power consumption than staying in the idle state. To model this, we define a *break-even time* ($T_{breakeven}$). The break-even time is the length of time that a node must be completely idle before it costs less in terms of power to turn it off than it does to leave it on. Empty nodes not needed for the global buffer remain on for $T_{breakeven}$ before PADD turns them off. The break-even time is defined as the period, for which a node continues to stay in the idle state before its power down so that we can achieve energy savings. In other words, the breakeven time is obtained when the cost is equal to the benefit in terms of energy consumption. When changing power state in a machine, there exists a breakeven time period that can achieve power savings. The break-even time of the node (n) is calculated as follows.

$$T^n_{breakeven} = \frac{(P^n_{O\rightarrow A} \times T_{O\rightarrow A} + P^n_{A\rightarrow O} \times T_{A\rightarrow O})}{P^n_A(0)}$$

4.3 Additional Parameters

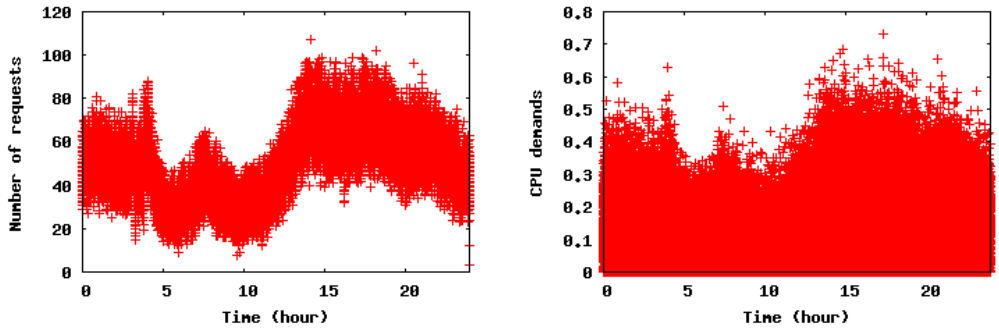
There are some other aspects of PADD which are also parameterized. These parameters refine the calculation of a domain's demand over time, the choice of domains to migrate and the assumed length of time that it takes to migrate a domain. These parameters are defined below.

1. Moving Window size (MW): size of the time window used to statistically summarize demands
2. Average Percentile (AP): percentile (%) to be ignored in the average of the demand. This can remove outliers in the demand data in order to avoid under-provisioning of system capacity.
3. Victim Selection (VS): how to choose a domain to be evicted when the remaining capacity in a node is less than LB. There are three (3) victim selection schemes are defined: maximum average demand, minimum average demand, and minimum standard deviation of demands.
4. Migration Delay (MD): time delay (sec) between migrations. This is used to avoid frequent migrations.

4.4 Algorithm

In Algorithm 1, we give the pseudo-code for the PADD algorithm. Periodically, the algorithm runs to adjust overall capacity and reduce power consumption. We use fractional units of a CPU for the metrics of both system capacity and domain demand. We calculate utilization by dividing the demand by the capacity.

The algorithm responds to the current demand by determining how many nodes to use and doing any migrations required. There may be SLA violations due to the fact that PADD is using too few nodes as the



(a) Access patterns (b) CPU Demands

Figure 3. One day access logs of the sporting event web site and synthetically generated CPU utilizations for a single domain.

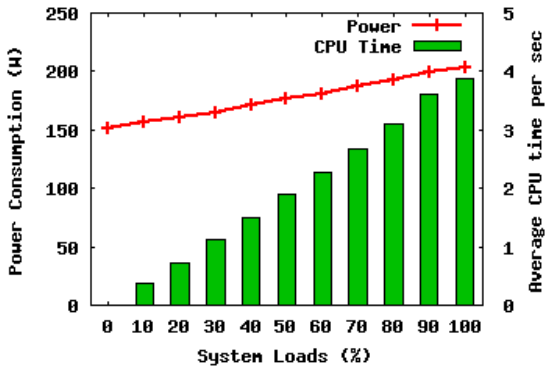


Figure 4. Average power consumption and CPU utilizations for 11 different loads from SPECpower_ssj2008 benchmark.

demand rises. We also simulate the performance effects of the migration. The *Premigration* state has no performance impact, but in the migration state, all the requested demands are delayed until it leaves the state. This may results in some SLA violations. The PADD simulation keeps track of these delays and makes a detailed analysis of SLA violations.

When determining a victim node due to over-provisioning (line 23), we find nodes on the three condition: no move in/out domain, no SLA violation in running domains, and idle for more than $T_{breakeven}$ time defined in Sec 4.2. If there is under-provisioning (line 35), on the other side, we increase the number of active nodes. If heterogeneous nodes are given, we will select the node in the order of high power-performance efficiency.

5 Evaluation

5.1 Workload generation

In this paper, we use traces obtained from the web site logs for a major international sporting event. Figure 3(a) shows the access patterns of the web site for a day. The y-axis represents the number of requests. The request patterns show that there exists significant variability in both daily and hourly accesses.

Param	Value	Param	Value
N	40	M	40
P_O	0 (W)	P_S	110 (W)
$P_{A\rightarrow S}$	151 (W)	$P_{S\rightarrow A}$	151 (W)
$P_{S\rightarrow O}$	187 (W)	$P_{O\rightarrow S}$	187 (W)
$T_{A\rightarrow S}$	1.0 (s)	$T_{S\rightarrow A}$	1.0 (s)
$T_{S\rightarrow O}$	30 (s)	$T_{O\rightarrow S}$	180 (s)
T_P	60 (s)	T_M	3 (s)
C	4.0	SLA	SLA(99,5),SLA(95,1)

Table 2. Parameter values for the simulation.

While we may be using traces derived from a web-based workload, it is important to note that we are simulating long-lived, stateful applications. We use a web-derived trace in the absence of direct measurements of such applications; this is a reasonable technique, because long-lived back-end applications are often driven by a transient web-based request stream, so they share the same usage pattern.

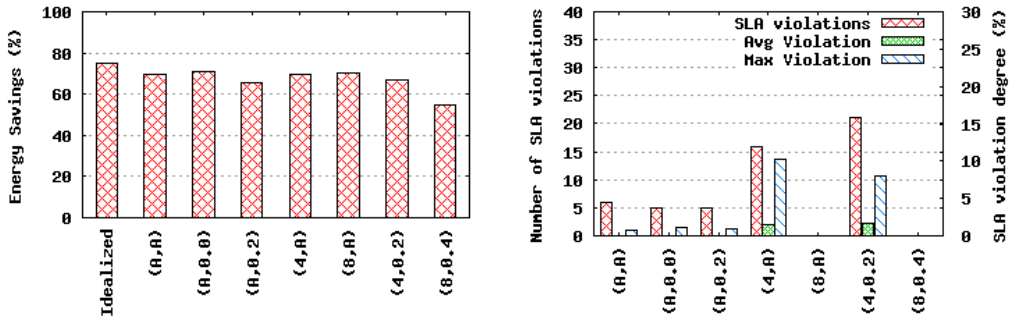
Since we do not have access to measured CPU utilizations, we generate synthetic CPU demands from the access traces by running the SPECpower_ssj2008 benchmark on a real system. We transform the request trace into a set of CPU utilizations using a calibration procedure that is based on measured workload intensity and measured CPU utilization for a representative program. The intensity levels used with the program range from 100% of maximum possible load to 0% in increments of 10%. We count the transactions processed while measuring the CPU utilization, and then, based on these data, we do a linear interpolation to calculate for a particular request rate in the trace, the associated CPU utilization. We acknowledge that this is not ideal. But it is the best available approach. The benchmark generates throughput-oriented workloads and provides precise throughput control based on a measured maximum.

The system used in the generation of the CPU utilization is an AMD Opteron dual-core, dual-processor machine with 4GB memory. While running the benchmark, we obtain CPU utilization and power draw for the different load levels as shown in Figure 4. Since the machine has 4 CPU cores, the total CPU time per second is always less than or equal to 4.0. The power consumption ranges between 151 and 203 watts. Based on this, we generate a synthetic CPU demand trace by applying a normal probability distribution. We also scale the number of requests in the traces to generate a more representative set of processor utilizations. Figure 3(b) shows the CPU demands generated with the access logs from Figure 3(a). In order to generate multiple CPU demands while maintaining the pattern, we amplified the coarse-grain trace. Multiple days of CPU demands are generated in the same way to simulate many domains running.

5.2 System configuration

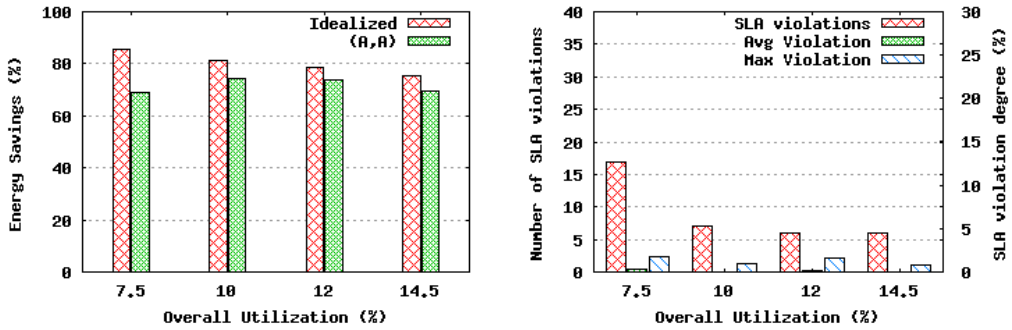
Table 2 shows time delays and the power usage for all node state transitions, as measured on our hardware. These server characteristics are used as input to all the simulations.

In our evaluation, we simulate 40 domains, where each domain's trace is taken from a different day's access pattern. The samples in the trace are 1 second apart. We assume that the CPU demand in a domain can require up to 1.0 CPU per second. As a performance requirement, we impose two SLAs. Half of the domains must process 99% of the incoming requests in 5 seconds while the other half must process 95% of them in 1 second. In this paper, we simulate 40 physical nodes with the same capacity. We do not consider the case where the capacity (C) is different by nodes. This is in the highest priority in our future work.



(a) Energy savings of various policies (b) Performance impact of various policies

Figure 5. The overall comparison of the representative cases. (Util=14.5%, MW=60, AP=60, VS=1, MD=30)



(a) Energy savings for varying workload intensity (b) Performance impact of (A, A) for varying workload intensity

Figure 6. The comparison of the adaptive GB/LB with different workload intensity. (MW=60, AP=60, VS=1, MD=30)

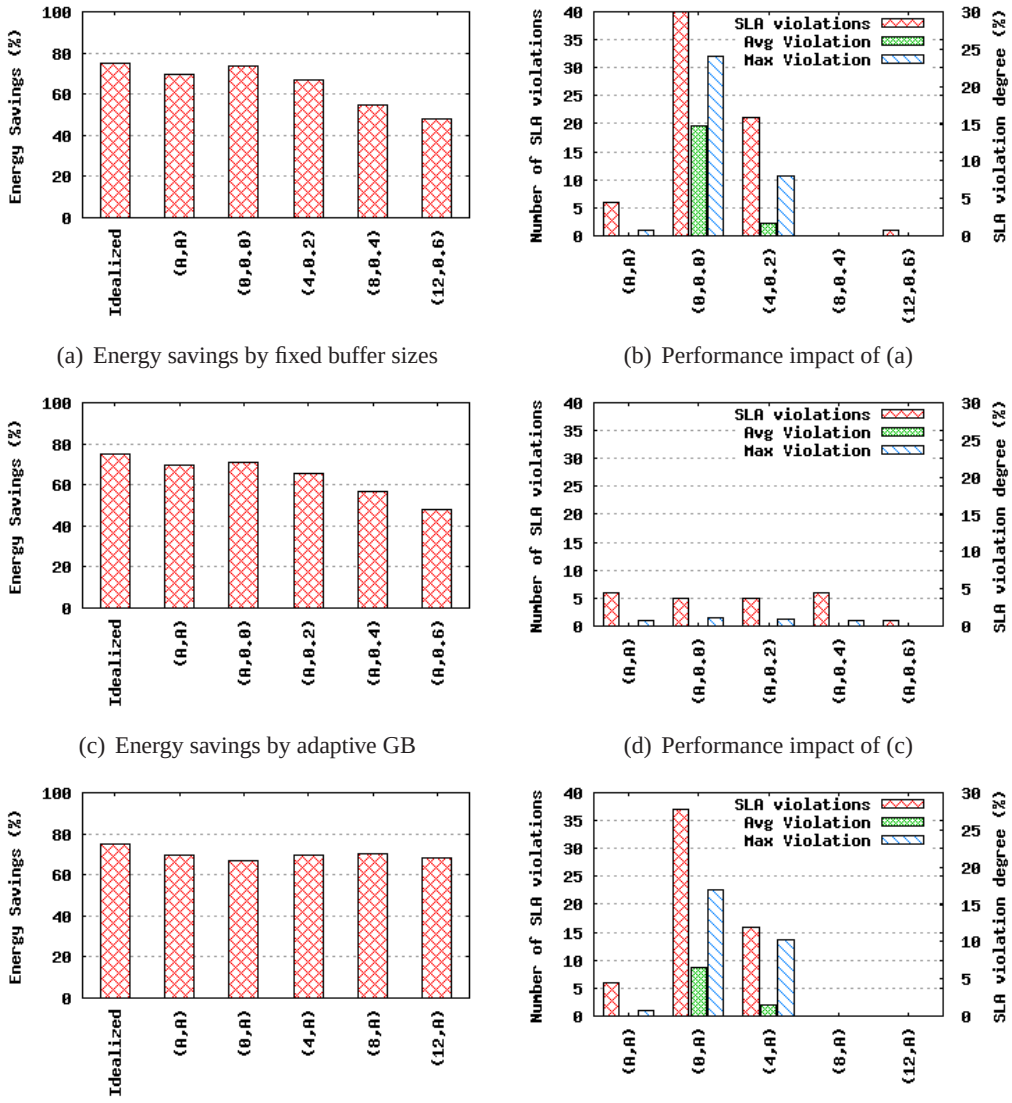
5.3 Analysis

Next, we analyze the energy savings and the performance impact caused by different configurations. The energy savings are calculated by comparing against a base case in which there was no consolidation. We measure performance in terms of how well SLAs were respected. To this end, we define the *violation degree (D_i)* as the difference between the desired success rate and the measured rate in the domain i. For example, if a domain has SLA(95,1), then it needs to meet 95% of demands within one second. If only 93% of demands are met in the given time, then the violation degree is 2% ($95 - 93$). When considering multiple domains, we present either the *average violation degree* ($\frac{\sum_i^M D_i}{M}$) or the *maximum violation degree* ($max(D_i)$).

For each evaluation, we compare 5 different PADD approaches while fixing some parameter values.

1. *Idealized*: there are no migration and transition time delays. GB and LB are not necessary. The *Idealized* always gives the lowest energy consumption without performance hurt and represents the best tradeoffs.

2. *Adaptive GB and LB (A,A)*: GB and LB are dynamically determined depending on SLA violation at



(a) Energy savings by fixed buffer sizes

(b) Performance impact of (a)

(c) Energy savings by adaptive GB

(d) Performance impact of (c)

(e) Energy savings by adaptive LB

(f) Performance impact of (e)

Figure 7. The comparison of adaptive buffering and fixed buffering. (Util=14.5%, MW=60, AP=60, VS=1, MD=30)

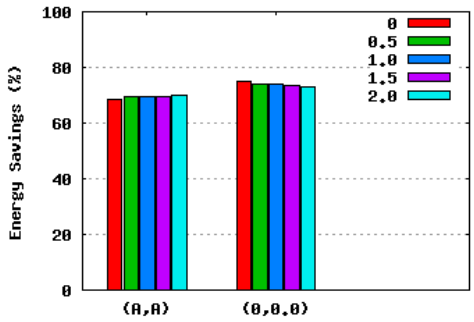
runtime.

3. *Adaptive GB and fixed LB (A,l)*: only GB is dynamically determined. The LB is fixed to l.

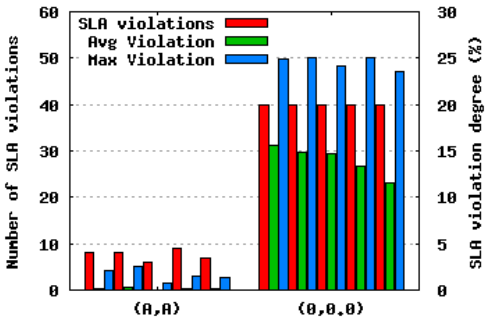
4. *Fixed GB and adaptive LB (g,A)*: only LB is dynamically determined. The GB is fixed to g.

5. *Fixed GB and LB (g,l)*: During simulation, GB and LB are the fixed values g and l, respectively.

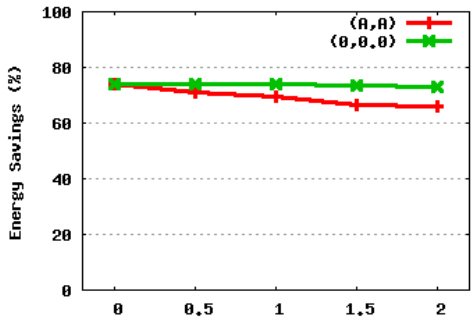
Figure 5 shows the overall energy savings and performance impact of the representative cases. The best case is $(8, A)$ which indicates an adaptive LB with a fixed GB of 8.0 CPUs (2 nodes). This case does not have any SLA violations. The fully adaptive (A, A) case does nearly as well, achieving 92.5% of the



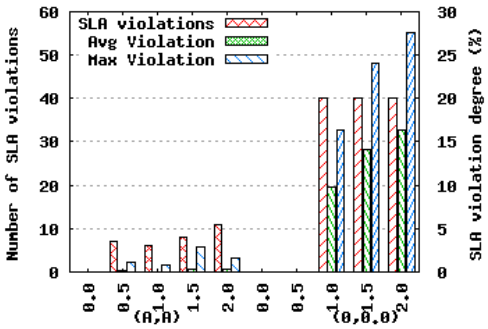
(a) Energy savings by different node transition costs. Migration cost is set to 1.0.



(b) Performance impact of (a)

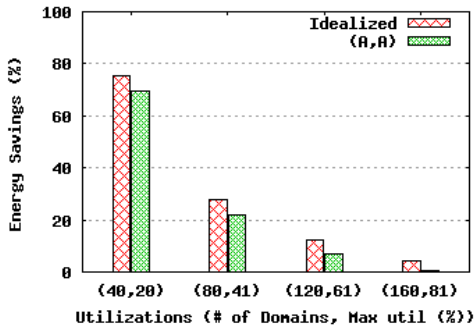


(c) Energy savings by different domain migration costs. Transition cost is set to 1.0.

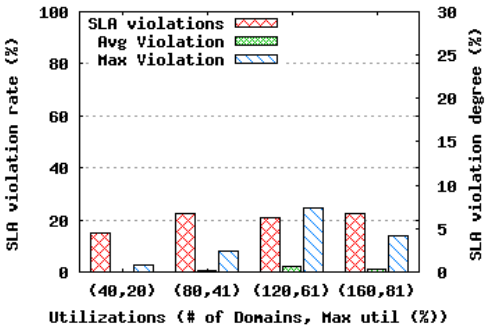


(d) Performance impact of (a)

Figure 8. The comparison of domain migration cost. (Util=14.5%, MW=60, AP=60, VS=1, MD=30)



(a) Energy savings by varying the number of domains.



(b) Performance impact of (A,A) for different number of domains

Figure 9. The comparison of the adaptive GB/LB with different number of domains. On X-axis, (x,y) indicates the number of domains and the maximum aggregate utilization, respectively. (MW=60, AP=60, VS=1, MD=30)

energy savings of the *Idealized* system with almost no average violation degree. With fixed GB/LB, $(4, 0.2)$ achieves similar energy savings to (A, A) but it has a maximum violation degree of 10%. The $(8, 0.4)$ configuration over-provisions capacity and therefore achieves significantly lower energy savings.

In Figure 6, we show the energy savings of the adaptive GB/LB approach with respect to varying workload intensity. The average utilization is formulated as the sum of demands of all domains for simulation time divided by total CPU capacity provided. As the utilization increases, the energy savings of the system decreases by about 10%. The adaptive GB/LB approach achieves 90% of the energy savings of the *Idealized* system on average. At a utilization of 7.5%, there are more opportunities to consolidate domains, which causes more node transitions and domain migrations. This results in more energy savings but more SLA violations. The average and maximum violation degrees, however, are limited to 0.39% and 1.79%, respectively.

In Figure 7, we evaluate the buffering schemes: fixed GB/LB, adaptive GB, adaptive LB, and adaptive GB/LB. In particular, (A,0) or (0,A) can be regarded as one level adaptive buffering scheme.

Overall, the adaptive GB/LB outperforms the others by achieving energy savings within 7% of the *Idealized* system with limited performance penalty. Various fixed GB/LB configurations are shown in Figures 7(a) and 7(b). In this case, only $(8, 0.4)$ and $(12, 0.6)$ offer acceptable performance. With the adaptive GB configurations shown in Figures 7(c) and 7(d), we see that the choice of LB has little effect on the performance impact but a major effect on energy savings. The reason is that the role of LB is taken by GB. Once enough GB is given, performance is not likely hurt much by long node transition time. In fact, the increase of LB in the adaptive GB approach reduces energy savings by 23%. In the adaptive LB approach (Figures 7(e) and 7(f)), the reverse is true: varying GB has a much greater effect on performance than energy savings. This is because as GB approaches zero, migrations from overloaded servers are having to wait for compute resources to come online.

In Figure 8, we show how the domain migration cost affects energy savings and performance. Each time cost is normalized to the real measured time delay specified in Table 2. When increasing the domain migration time while transition delay is fixed, the energy savings in the adaptive GB/LB is reduced (Figure 8(c)). This is because more processing delays occurred followed by more domains involved in transitions. This performance penalty is clearly shown in the case $(0, 0.0)$ in Figure 8(d). When the domain migration time becomes 0, there is almost no performance penalty for migration and the greatest energy savings is achieved. The only other potential source of performance degradation is unexpected variation of CPU demand, which we account for not only with the Global and Local Buffers, but also average percentile (60% in this case). Therefore, the given SLA requirements are generally met, although there can be some delays due to the transition time (e.g. when a busy node is waiting for additional nodes to come online).

For reasons of space, we omit the figures regarding the effects of node transition times as well as some other parameters of PADD. As node transition time increases while the migration cost is fixed at 1.0, we miss out on some power saving opportunities, resulting in diminished energy savings. However, the difference between 0.0 and 2.0 is negligible (less than 3%). In terms of performance impact, for fixed LB/GB, the increasing node transition cost actually has a positive effect, because it blocks unnecessary domain migration caused by transient spikes in demand. In our adaptive GB/LB approach, node transition cost does not significantly affect the overall performance. We also evaluated different policies with changing the parameters defined before. Our findings from those changes are less interesting. We did not find that the changes had much of an impact, compared with GB/LB sizes and management.

In Figure 9, we show the energy savings and performance impacts of the adaptive GB/LB approach by increasing the number of domains. The same 40 demand traces are used for additional domains. Accordingly, the overall utilization increases as the number of domains increases. On the x-axis, (x,y) represents the number of domains (x) and the maximum utilization (y) respectively. As the number of domains increases along with overall utilization, the energy savings in the *Idealized* system decreases from 75.2% to 4.3%, due to reduced residual in processing capacity. Our adaptive scheme achieves energy savings within 6% of the

Idealized system. The SLA violation rate ($\frac{Number\ of\ SLA\ Violations}{Number\ of\ Domains}$), however, remains below 22.5%. The average violation degree is 0.6% or less in all cases, although additional migrations caused by running more domains increase the maximum violation degree up to 7.4%.

6 Conclusion

This paper presents an algorithm for consolidating domains in a virtualized environment running coarse-grained workloads to reduce overall energy consumption. We show through a simulation-based study that we get the best combination of energy savings and performance by using an adaptive buffering scheme to determine how much reserve capacity is needed. We use a two-level scheme with a Local Buffer on each node and an additional Global Buffer. A number of other dimensions were evaluated including the time to migrate a domain and the time to change the power state of a node, but none of them have as much effect on PADD as the size and management of the buffers. The best trade-off for effectively no performance loss and maximum energy savings is the adaptive GB/LB approach. More results are presented in [17].

Our future work is twofold. First, we want to run our simulator using traces from a wider variety of workloads and incorporate mixes of different workloads. We believe that mixing different workloads in the simulation is more realistic for emerging computing environments such as utility grids and clouds. Second, we made a number of simplifying assumptions which make our pool of nodes homogeneous. This is clearly unrealistic for any large installation. We need to extend our algorithm and our simulation to consider machines with different characteristics.

In the long term, the simulator will provide an understanding of the design principles involved in the PADD system, allowing us to develop a practical implementation.

References

[1] Elastic Compute Cloud (Amazon EC2). http://aws.amazon.com/ec2/.

[2] IBM Blue Cloud Computing. http://www-03.ibm.com/press/us/en/pressrelease/22613.wss.

[3] Virtual Computing Lab. http://vcl.ncsu.edu/.

[4] VMware ESX Server. http://www.vmware.com/products/vi/esx/.

[5] Xen Virtualization Technology. http://www.xen.org/.

[6] M. Banikazemi, D. Poff, and B. Abali. PAM: A Novel Performance/Power Aware Meta-scheduler for Multi-core Systems. In *Supercomputing conference (SC)*, 2008.

[7] N. Bobroff, A. Kochut, and K. Beaty. Dynamic Placement of Virtual Machines for Managing SLA Violations. *Integrated Network Management, 2007. IM '07. 10th IFIP/IEEE International Symposium on*, pages 119–128, 21 2007-Yearly 25 2007.

[8] D. J. Bradley, R. E. Harper, and S. W. Hunter. Workload-based power management for parallel computer systems. In *IBM Journal of Research and Development*, 2003.

[9] J. S. Chase, D. C. Anderson, P. N. Thakar, and A. M. Vahdat. Managing Energy and Server Resources in Hosting Centers. In *the Eighteenth ACM Symposium on Operating Systems Principles*, 2001.

[10] G. Chen, W. He, J. Liu, S. Nath, L. Rigas, L. Xiao, and F. Zhao. Energy-Aware Server Provisioning and Load Dispatching for Connection-Intensive Internet Services. In *5th Symposium on Networked Systems Design USENIX Association and Implementation (NSDI)*, 2008.

[11] C. Clark, K. Fraser, S. Hand, J. G. Hansen, E. Jul, C. Limpach, I. Pratt, and A. Warfield. Live Migration of Virtual Machines. In *2nd Symposium on Networked Systems Design and Implementation*, 2005.

[12] U. Drepper. The Cost of Virtualization. *ACM Queue*, 2008.

[13] E. N. Elnozahy, M. Kistler, and R. Rajamony. Energy-Efficient Server Clusters. In *2nd Workshop on Power-Aware Computing Systems*, 2002.

[14] V. W. Freeh, T. K. Bletsch, and F. L. Rawson. Scaling and Packing on a Chip Multiprocessor. In *Workshop on High-Performance, Power-Aware Computing*, 2007.

[15] V. W. Freeh, D. K. Lowenthal, F. Pan, and N. Kappiah. Using Multiple Energy Gears in MPI Programs on a Power-Scalable Cluster. In *Principles and Practices of Parallel Programming (PPOPP)*, 2005.

[16] S. Ghiasi and W. Felter. CPU Packing for Multiprocessor Power Reduction. In *Power Aware Computer Systems (PACS)*, 2003.

[17] M. Y. Lim, T. Bletsch, V. W. Freeh, and F. Rawson. PADD: Power-Aware Domain Distribution. In *Technical Report To be assigned, IBM Research*, 2009.

[18] M. Y. Lim, V. W. Freeh, and D. K. Lowenthal. Adaptive, Transparent Frequency and Voltage Scaling of Communication Phases in MPI Programs. In *IEEE/ACM Supercomputing 2006*, 2006.

[19] R. Nathuji and K. Schwan. VirtualPower: Coordinated Power Management in Virtualized Enterprise Systems. In *ACM Symposium on Operating Systems Principles (SOSP)*, 2007.

[20] R. Nathuji and K. Schwan. VPM Tokens: Virtual Machine-Aware Power Budgeting in Datacenters. In *HPDC '08: Proceedings of the 17th international symposium on High performance distributed computing*. ACM, 2008.

[21] R. Neapolitan and K. Naimipour. *Foundations of Algorithms*. 1997.

[22] P. Padala, X. Zhu, Z. Wang, S. Singhal, and K. G. Shin. Performance Evaluation of Virtualization Technologies for Server Consolidation. Technical report, HP Lab., April 2007.

[23] K. Rajamani and C. Lefurgy. On Evaluating Request-Distribution Schemes for Saving Energy in Server Clusters. In *IEEE Performance Analysis of System and Software*, 2003.

[24] A. Singh, M. Korupolu, and D. Mohapatra. Server-Storage Virtualization: Integration and Load Balancing in Data Centers. In *Supercomputing conference (SC)*, 2008.

[25] M. Steinder, I. Whalley, D. Carrera, I. Gaweda, and D. Chess. Server virtualization in autonomic management of heterogeneous workloads. *Integrated Network Management, 2007. IM '07. 10th IFIP/IEEE International Symposium on*, 2007.

[26] A. Verma, P. Ahuja, and A. Neogi. pMapper: power and migration cost aware application placement in virtualized systems. In *Middleware '08: Proceedings of the 9th ACM/IFIP/USENIX International Conference on Middleware*. Springer-Verlag New York, Inc., 2008.

[27] A. Verma, P. Ahuja, and A. Neogi. Power-aware dynamic placement of HPC applications. In *ICS '08: Proceedings of the 22nd annual international conference on Supercomputing*. ACM, 2008.

[28] W. Vogels. Beyond Server Consolidation. *ACM Queue*, 2008.

Algorithm 1 Power Aware Domain Distribution

1: // Retrieve the node/domain states.
2: $updateNodeandDomainStates()$
3: // Find all the nodes where LB is below the specified level
4: **for** $n = 1$ to N **do**
5: **if** $C_n < LB_n + \sum Avg(D_i)$ **then**
6: $nodes \leftarrow n$
7: **end if**
8: **end for**
9: **for** n in $nodes$ **do**
10: // Choose a victim domain in the node. see Sec 4.3
11: $domainID \leftarrow selectVictimDomain(n)$
12: // Determine where to migrate the domain
13: $targetNodeID \leftarrow findTargetNode(domainID)$
14: // If there exists an active node that has sufficient LB
15: **if** $targetNodeID$ is valid **then**
16: $beginMigration(domainID, targetNodeID)$
17: **end if**
18: **end for**
19: // Check if there is enough provisioning in GB and LB
20: $reqCap \leftarrow GB + \sum_{i=1}^{N} LB_i + \sum_{j=1}^{D} Avg(D_j) - \sum^{NS_i=Active} C_i$
21: **if** $reqCap < 0$ **then**
22: // If there is over-provisioning in total system capacity,
23: // find the nodes where domains can be consolidated
24: $nodelist = getTurnOffCandidates()$
25: **for** $nodeID$ in $nodelist$ **do**
26: **if** $reqCap + C_{nodeID} < 0$ **then**
27: // If all the domains in the node can fit into other nodes,
28: // vacate the node by evicting all of them,
29: **if** $beginDomainConsolidation(nodeID)$ **then**
30: $reqCap \leftarrow reqCap + C_{nodeID}$
31: **end if**
32: **end if**
33: **end for**
34: **else if** $reqCap > 0$ **then**
35: **repeat**
36: $nodeID = getNodeWithHighestEfficiency()$
37: $powerupNode()$
38: $reqCap \leftarrow reqCap - C_{nodeID}$
39: **until** $reqCap \leq 0$
40: **end if**