vGreen: A System for Energy-Efficient Management of Virtual Machines

GAURAV DHIMAN, GIACOMO MARCHETTI, and TAJANA ROSING
University of California, San Diego

In this article, we present vGreen, a multitiered software system for energy-efficient virtual machine management in a clustered virtualized environment. The system leverages the use of novel hierarchical metrics that work across the different abstractions in a virtualized environment to capture power and performance characteristics of both the virtual and physical machines. These characteristics are then used to implement policies for scheduling and power management of virtual machines across the cluster. We show through real implementation of the system on a state-of-the-art testbed of server machines that vGreen improves both average performance and system-level energy savings by close to 40% across benchmarks with varying characteristics.

Categories and Subject Descriptors: D.4.7 [**Operating Systems**]: Distributed Systems

General Terms: Design, Experimentation, Performance

Additional Key Words and Phrases: Virtualization, migration, energy, workload characterization

ACM Reference Format:
Dhiman, G., Marchetti, G., and Rosing, T. 2010. vGreen: A system for energy-efficient management of virtual machines. ACM Trans. Des. Autom. Electron. Syst. 16, 1, Article 6 (November 2010), 27 pages. DOI = 10.1145/1870109.1870115. http://doi.acm.org/10.1145/1870109.1870115.

1. INTRODUCTION

Power consumption is a critical design parameter in modern data center and enterprise environments, since it directly impacts both the deployment (peak power delivery capacity) and operational costs (power supply, cooling). The energy consumption of the compute equipment and the associated cooling infrastructure is a major component of these costs. The electricity consumption for powering the data centers in the U.S. is projected to cross $7B by the end of

A preliminary version of this article appeared in ISLPED 2009 [Dhiman et al. 2009].
This work has been funded in part by Oracle Corp., UC MICRO, Center for Networked Systems (CNS) at UCSD, MARCO/DARPA Gigascale Systems Research Center and NSF Greenlight
Authors' address: G. Dhiman (corresponding author), G. Marchetti, T. Rosing, Department of Computer Science, University of California, San Diego, CA; email: gdhiman@cs.ucsd.edu.

2010 [Meisner et al. 2009; Pakbaznia and Pedram 2009]. This provides strong motivation for developing mechanisms to efficiently manage computation in data centers.

Modern data centers and cloud computing providers (like Amazon EC2 [Amazon 2008]) use virtualization (e.g., Xen [Barham et al. 2003] and VMware [Haletky 2008]) to get better fault and performance isolation, improved system manageability, and reduced infrastructure cost through resource consolidation and live migration [Clark et al. 2005]. Consolidating multiple servers running in different Virtual Machines (VMs) on a single Physical Machine (PM) increases the overall utilization and efficiency of the equipment across the whole deployment. Thus, the creation, management, and scheduling of VMs across a cluster of PMs in a power-aware fashion is key to reducing the overall operational costs. Policies for power-aware VM management have been proposed in previous research [Raghavendra et al. 2008] and are available as commercial products as well (e.g., VMware DRS [VMware 2009]). These policies require understanding of the power consumption and resource utilization of the PM, as well as its breakdown among the constituent VMs for optimal decision making. Currently they treat the overall CPU utilization of the PM and its VMs as an indicator of their respective power consumption and resource utilization, and use it for guiding the VM management policy decisions (VM migration, dynamic voltage frequency scaling or DVFS, etc.). However, in our work we show that based on the characteristics of these different colocated VMs, the overall power consumption and performance of the VMs can vary a lot even at similar CPU utilization levels. This can mislead the VM management policies into making decisions that can create hotspots of activity, and degrade overall performance and energy efficiency.

In this article, we introduce vGreen, a multitiered software system to manage VM scheduling across different PMs with the objective of managing the overall energy efficiency and performance. The basic premise behind vGreen is to understand and exploit the relationship between the architectural characteristics of a VM (e.g., instructions per cycle, memory accesses, etc.) and its performance and power consumption. vGreen is based on a client server model, where a central server (referred to as "*vgserv*") performs the management (scheduling, DVFS, etc.) of VMs across the PMs (referred to as "*vgnodes*"). The *vgnodes* perform online characterization of the VMs running on them and regularly update the *vgserv* with this information. These updates allow *vgserv* to understand the performance and power profile of the different VMs and aids it to intelligently place them across the *vgnodes* to improve overall performance and energy efficiency.

We implemented vGreen on a testbed of state-of-the-art servers running Xen as the virtualization software (known as hypervisor). For evaluation, we created and allocated VMs across the PM cluster, which ran benchmarks with varying workload characteristics. We show that vGreen is able to dynamically characterize VMs, and accurately models their resource utilization levels. Based on these characteristics, it can intelligently manage the VMs across the PM cluster by issuing VM migration or DVFS commands. As we show in the latter sections, this improves the overall performance up to 100% and energy

efficiency up to 55% compared to state-of-the-art VM scheduling and power management policies. Moreover, vGreen is very lightweight with negligible runtime overhead.

The rest of the article is organized as follows: In Section 2, we discuss the related state-of-the-art work and outline our contributions. This is followed by a discussion on the background and motivation for the approach we adopt to solve the problem in Section 3. In Section 4, we explain the overall design, architecture and implementation details of the vGreen system. We provide details of the overall methodology for evaluation of our system and the experimental results in Section 5 before concluding in Section 6.

2. RELATED WORK

Systems for management of VMs across a cluster of PMs have been proposed in the past. Eucalyptus [Nurmi et al. 2008], OpenNebula [OpenNebula] and Usher [McNett et al. 2007] are open-source systems which include support for managing VM creation and allocation across a PM cluster. For management of VMs on larger scale, for instance across multiple data center sites, systems like Grid Virtualization Engine (GVE) [Wang et al. 2009] have been proposed. However, these solutions do not have VM scheduling policies to dynamically consolidate or redistribute VMs. VM scheduling policies for this purpose have also been investigated in the past. In Wood et al. [2007], the authors propose a VM scheduling system which dynamically schedules the VMs across the PMs based on their CPU, memory, and network utilization to avoid hotspots of activity on PMs for better overall performance. The Distributed Resource Scheduler (DRS) from VMware [VMware 2009] also uses VM scheduling to perform automated load balancing in response to CPU and memory pressure. Similarly, in Bobroff et al. [2007], the authors propose VM scheduling algorithms for dynamic consolidation and redistribution of VMs for managing performance and SLA (Service-Level Agreement) violations. In Wang and Kandasamy [2009], the authors model application performance across VMs to dynamically control the CPU allocation to each VM with the objective of maximizing the profits. The authors in Hermenier et al. [2009] propose Entropy, which uses constraint programming to determine a globally optimal solution for VM scheduling in contrast to the first-fit decreasing heuristic used by Wood et al. [2007] and Bobroff et al. [2007], which can result in globally suboptimal placement of VMs. However, none of these VM scheduling algorithms takes into account the impact of the policy decisions on the energy consumption in the system.

Power management in data-center-like environments has been an active area of research. In Chase et al. [2001], data center power consumption is managed by turning servers off depending on demand. Reducing operational costs by performing temperature-aware workload placement has also been explored [Moore et al. 2005]. In Ge et al. [2007], Dynamic Voltage Frequency Scaling (DVFS) is performed based on the memory intensiveness of workloads on the server clusters to reduce energy costs. Similarly, Ranganathan et al. [2006] and Fan et al. [2007] use DVFS to reduce average power consumption in blade servers with the objective of performing power budgeting. Recent studies

[Dhiman et al. 2008; Meisner et al. 2009] have shown that in modern server systems, the effectiveness of DVFS for energy management has diminished significantly due to its impact on the performance of the workloads. In this article, we confirm this observation and show how intelligent VM colocation outperforms state-of-the-art DVFS policies [Dhiman and Rosing 2007; Isci et al. 2006] in terms of energy savings.

The problem of power management in virtualized environments has also been investigated. In Nathuji and Schwan [2007], the authors propose VirtualPower, which uses the power management decisions of the guest OS on virtual power states as hints to run local and global policies across the PMs. It relies on efficient power management policies in the guest OS, and does no VM characterization at the hypervisor level. This makes it difficult to port some of the state-of-the-art power management policies like Dhiman and Rosing [2007] and Isci et al. [2006] in guest OS because of lack of exclusive access to privileged resources such as CPU performance counters. This problem has led to adoption of power management frameworks like *cpufreq* and *cpuidle* in recent virtualization solutions (like Xen [Barham et al. 2003]). In Abdelsalam et al [2009], the authors develop a power and performance model of a transaction-based application running within the VMs, and use it to drive cluster-level energy management through DVFS. However, they assume the application characteristics to be known. In Raghavendra et al. [2008], a coordinated multilevel solution for power management in data centers is proposed. Their solution is based on a model that uses power estimation (using CPU utilization) and overall utilization levels to drive VM placement and power management. However, the model and results are based on offline trace-driven analysis and simulations. In Liu et al. [2009], the authors present GreenCloud, an infrastructure to dynamically consolidate VMs based on CPU utilization to produce idle machines, which could be turned off to generate energy savings. However, none of these solutions [Abdelsalam et al 2009; Liu et al. 2009; Nathuji and Schwan 2007; Raghavendra et al. 2008] takes the architectural characteristics of the VM into account which, as we show in Section 3, directly determine the VM performance and power profile. In Verma et al. [2008], the authors use VM characteristics like cache footprint and working set to drive power-aware placement of VMs. But their study assumes an HPC application environment, where the VM characteristics are known in advance. Besides, their evaluation is based on simulations. In contrast, vGreen assumes a general-purpose workload setup with no apriori knowledge on their characteristics.

The concept of dynamic architectural characterization of workloads using CPU performance counters for power management [Dhiman and Rosing 2007; Isci et al. 2006], performance [Knauerhase et al. 2008], and thermal management [Merkel and Bellosa 2006] on nonvirtualized systems has been explored before. For a stand-alone virtualized PM, the authors in Stoess et al. [2007] use performance counters to enforce power budgets across VMs on that PM. In Merkel et al. [2010], the authors identify resource contention as a problem for energy efficiency, but primarily focus on scheduling and power management on a single PM. In some recent papers, performance counters have been used in virtualized clusters to perform power metering [Dhiman et al.

2010; Kansal et al. 2010; Koller et al. 2010], QoS management [Nathuji et al. 2010], and power budgeting [Nathuji et al. 2009] to aid efficient power provisioning. However, using architectural characterization to drive cluster-level VM management from the perspective of energy efficiency and balanced power consumption has been largely unexplored.

Based on this discussion, the primary contributions of our work are as follows: (1) We propose a system for characterization of VMs in a virtualized environment. It is based on novel hierarchical metrics that capture power and performance profiles of the VMs. (2) We use the online VM characterization to drive dynamic VM management across a PM cluster for overall performance and energy efficiency. (3) We implement the proposed system on a real-life testbed of state-of-the-art machines and through extensive set of experiments and measurements across machines with different architectures and configurations, and highlight the benefits of the approach over existing state-of-the-art approaches.

3. MOTIVATION

3.1 Background

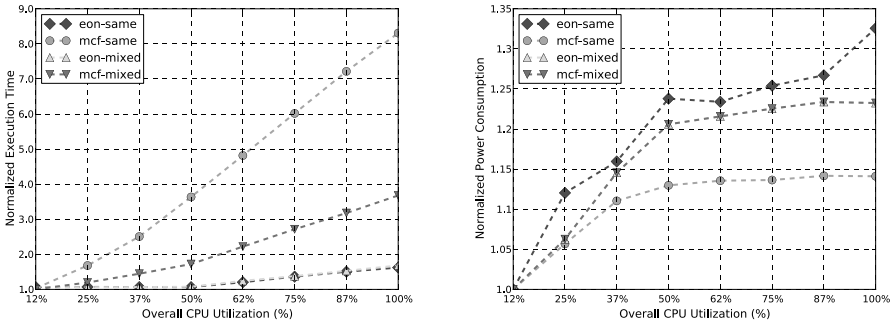
In this article we assume Xen as the underlying virtualization hypervisor. It is a standard open-source virtualization solution which also forms the baseline technology for commercial products like XenSource, Oracle VM, etc. However, the ideas presented in this work are independent of Xen, and can be applied to other virtualization solutions like Kernel-based Virtual Machines (KVM), etc., as well. In Xen, a VM is an instance of an OS which is configured with Virtual CPUs (VCPUs) and a memory size. The number of VCPUs and memory size is configured at the time of VM creation. Xen virtualizes the real hardware to the VM making the OS running within it believe that it is running on a real machine. A PM can have multiple VMs active on it at any point in time, and Xen multiplexes them across the real Physical CPUs (PCPUs) and memory. The entity that Xen schedules over the PCPU is the VCPU, making it the fundamental unit of execution. Thus, a VCPU is analogous to a thread, and a VM is analogous to a process in a system running a single OS like Linux. In addition, Xen provides a control VM, referred to as Domain-0 (or Dom-0), which is what the machine running Xen boots into. It acts as an administrative interface for the user, and provides access to privileged operations like creating, destroying, or migrating VMs.

3.2 Performance and Power Profile of VMs

The nature of workload executed in each VM determines the power profile and performance of the VM, and hence its energy consumption. As discussed before, VMs with different or same characteristics could be colocated on the same PM. In this section, we show that colocation of VMs with heterogeneous characteristics on PMs is beneficial for overall performance and energy efficiency across the PM cluster.

For understanding this, we performed some experiments and analysis on two benchmarks from SPEC-CPU 2000 suite, namely *eon* and *mcf*. These two benchmarks have contrasting characteristics in terms of their CPU and memory utilization. While *mcf* has high Memory Per Cycle (MPC) accesses and low Instructions committed Per Cycle (IPC), *eon* has low MPC and high IPC. We use a testbed of two dual Intel quad core Xeon (hyper-threading equipped)-based PMs (sixteen CPUs each) running Xen. On each of these PMs, we create two VMs with eight virtual CPUs (VCPUs) each (total of four VMs). Inside each VM we execute either *eon* or *mcf* as the workload. We use multiple instances/threads of the benchmarks to generate higher utilization levels. For our PM (sixteen CPUs), this implies four instances for 25% utilization, eight instances for 50%, and sixteen instances for 100% utilization. Each PM is equipped with power sensors which are interfaced to the Dom-0 OS in a standardized fashion using Intelligent Platform Management Interface (IPMI) [IPMI 2004]. We periodically (every 2s) query the IPMI interface to log the power consumption of the whole PM for all our experiments.

In our first set of experiments, we run homogeneous VMs on each PM, that is, the two VMs with *mcf* on one PM, and two with *eon* on the other. We refer to this VM placement schedule as "same" indicating homogeneity. During the execution, we record the execution time of all the benchmark instances. Figure 1(a) shows the normalized execution time results for different number of instances of the benchmarks, where the execution times are normalized against the execution time with two instances (one instance per VM). We can observe that for *mcf* in the "same" schedule (shown as "mcf-same"), as the CPU utilization increases, the execution time almost increases linearly. For 100% utilization *mcf*, the execution time is almost 8.5x compared to the baseline execution time. The primary reason for such an observation is the high MPC of *mcf*. The high MPC results in higher cache conflict rate and pressure on the memory bandwidth when multiple threads execute, which decreases the effective IPC per thread and hence increases its execution time. This is illustrated by the plot of aggregate IPC and MPC of all *mcf* threads in Figure 1(c). We can see how the MPC increases by around 7x as CPU utilization goes from 12% to 100%. However, the aggregate IPC almost remains constant, which implies IPC per thread goes down significantly, resulting in increased execution time observed in Figure 1(a). In contrast, for *eon* ("eon-same"), the execution time is fairly independent of the CPU utilization due to its much lower MPC. We can observe that the execution time shows an increase beyond 50% utilization. This happens since our machine has eight cores and sixteen CPUs (due to hyper-threading), with two CPUs per core. When we reach 50% utilization that corresponds to eight threads of the benchmark, and beyond that the threads start sharing the pipeline, which reduces the individual IPC of threads sharing the pipeline. This phenomena is illustrated in Figure 1(c), where the IPC slope of *eon* drops off a little beyond 50% CPU utilization. However, this increase in execution time is trivial compared to that of *mcf* as seen in Figures 1(a) and 1(c). In summary, this analysis indicates that the performance of a VM has a strong negative correlation to utilization rate of the memory subsystem.



(a) normalized execution time

(b) normalized power consumption

(c) comparison of aggregate IPC and MPC

(d) comparison of "mixed" vs. "same" VM placement schedules

Fig. 1. Comparison of *eon* and *mcf*.

Similarly, Figure 1(b) shows the system-level power consumption of the PMs normalized against the power consumption with just two threads. We can observe that for *eon* ("eon-same"), the power consumption increases almost linearly to the increase in utilization. This happens since it has high IPC, which implies higher CPU resource utilization and power consumption. We can observe that the slope of increase in power changes at 50% utilization. This is again due to pipeline sharing between threads beyond 50% utilization, which lowers the contribution of new threads to power consumption (see Figure 1(c)). In contrast, for *mcf*, the power consumption increases initially but then it saturates. This primarily happens due to the lower IPC of threads at higher utilization levels as discussed before. As a consequence of this, the difference in power consumption between the two PMs is almost 20% (∼45 Watts in our measurements). This analysis indicates that the power consumption of a VM has direct corelation to IPC of the workload running inside it.

These results indicate that coscheduling VMs with similar characteristics is not beneficial from the energy-efficiency point of view at the cluster level. The PM running *mcf* contributes to higher system energy consumption, since it runs for a significantly longer period of time. To understand the benefits of coscheduling heterogeneous workloads in this context, we swapped two VMs

on the PMs, hence running VMs with *mcf* and *eon* on both the PMs. We refer to this VM placement schedule as "mixed", indicating the heterogeneity. Figure 1 shows the results (indicated as "mixed") achieved for this configuration in terms of normalized execution time and power consumption. We can observe that *eon* execution time almost stays the same, while *mcf* execution time goes down significantly at higher utilization rates (around 450% reduction at 100% utilization). This happens because we now get rid of the hotspot of intense activity in the memory subsystem on one PM (running just the *mcf* VMs in the "same" schedule), and share the overall system resources in a much more efficient fashion. The average power consumption of the two PMs becomes similar, and roughly lies between that of the two PMs in the "same" schedule, as the overall IPC is also much better balanced across the cluster.

Figure 1(d) illustrates the comparison of the "mixed" and "same" VM schedules, and highlights the benefits of the "mixed" schedule. It plots three key metrics to capture this.

(1) *Energy savings*. We estimate the energy reduction in executing each combination of VMs using "mixed" over "same" schedule. This is calculated by measuring the total energy consumption for a VM combination with two schedules, and then taking their difference. We can observe that across all utilization levels, the "mixed" schedule is clearly more energy efficient compared to the "same" schedule. At higher utilization rates (50% and beyond), it achieves as high as 50% energy savings. This primarily happens due to the high speedup achieved by it compared to "same" schedule, as discussed before, while keeping the average power consumption almost similar. The next two metrics provides details on these.

(2) *Average Weighted Speedup (AWS)*. This metric captures how fast the workload runs on the "mixed" schedule compared to "same" schedule. The AWS is based on a similar metric defined in Snavely and Tullsen [2000]. It is defined as

$$AWS = \frac{\sum_{VM_i} \frac{T_{same_i}}{T_{alone_i}}}{\sum_{VM_i} \frac{T_{mixed_i}}{T_{alone_i}}} - 1, \tag{1}$$

where T_{alone_i} is the execution time of VM_i when it runs alone on a PM, and T_{same_i} and T_{mixed_i} are its execution time as part of a VM combination with "same" and "mixed" schedules, respectively. To calculate AWS, we normalize T_{same_i} and T_{mixed_i} against T_{alone_i} for each VM, and then take ratio of the sum of these normalized times across all the VMs in the combination as shown in Eq. (1). AWS > 0 implies that the VM combination runs faster with "mixed" schedule and vice versa. Figure 1(d) clearly shows that the "mixed" schedule is able to achieve significant speedup. The AWS reaches as high as 57% due to efficient resource sharing and contributes significantly to the energy savings discussed earlier.

(3) *Increase in power consumption*. This metric captures the difference between the average power consumption of the PMs under the "mixed" and "same" schedule. This is important, since we need to make sure that the

speedup achieved does not result in much higher average power consumption across the cluster. Figure 1(d) shows that the increase in system power consumption is trivial (<3%) across all the utilization levels. Thus, high speedups at almost similar average power consumption results in significant energy savings illustrated in Figure 1(d).

In summary, this discussion provides us key insights into the VM management problem: (1) VM characteristics provide invaluable information on both the power as well as performance profile of VMs. (2) VM scheduling policies should try to coschedule VMs with heterogeneous characteristics on the same PM. This results in efficient sharing of resources across the cluster and as a consequence is beneficial from both energy-efficiency and performance point of view. This is achievable in virtualized environments, since VMs can be dynamically migrated at runtime across PMs at low overhead using "*live migration*" [Clark et al. 2005].

This provides strong motivation to use online characterization of VMs for system-wide VM management. In the next section, we describe the overall architecture of vGreen, and present details on how it constructs VM characteristics dynamically at runtime using a novel hierarchical approach.

4. DESIGN

In this section, we present the details on the design, architecture, and implementation of our system, vGreen. Building upon the discussion in the last section, we show how the system is structured to capture the CPU and memory utilization rates of individual VMs, and how it uses it to manage the VMs in an efficient fashion across a PM cluster.

4.1 Architecture

Figure 2 illustrates the overall architecture of vGreen, which is based on a client-server model. Each PM in the cluster is referred to as a vGreen client/node (*vgnode*). There is one central vGreen server (*vgserv*) which manages VM scheduling across the *vgnodes* based on a policy (*vgpolicy*) running on the *vgserv*. The *vgpolicy* decisions are based on the value of different metrics, which capture MPC, IPC, and utilization of different VMs, that it receives as updates from the *vgnodes* running those VMs. The metrics are evaluated and updated dynamically by the vGreen modules in Xen (*vgxen*) and Dom-0 (*vgdom*) on each *vgnode*. Regular updates from the *vgnodes* on the metrics allow the *vgpolicy* to balance both the power consumption and overall performance across the PMs. We now describe the vGreen components and the metrics employed in detail.

4.1.1 *vgnode.* A *vgnode* refers to an individual PM in the cluster. A *vgnode* might have multiple VMs running on it at any given point in time as shown in Figure 2. Each *vgnode* has vGreen modules (*vgxen* and *vgdom*) installed on them.

vgxen. The *vgxen* is a module compiled into Xen (see Figure 2) and is responsible for characterizing the CPU and memory behavior (specifically IPC

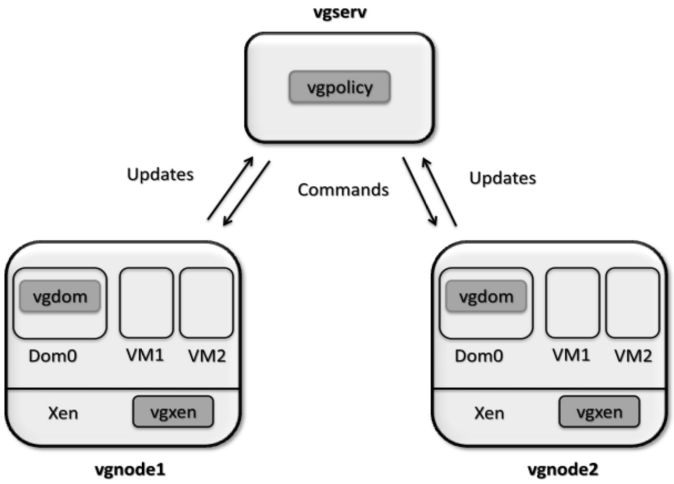


Fig. 2. Overall vGreen design.

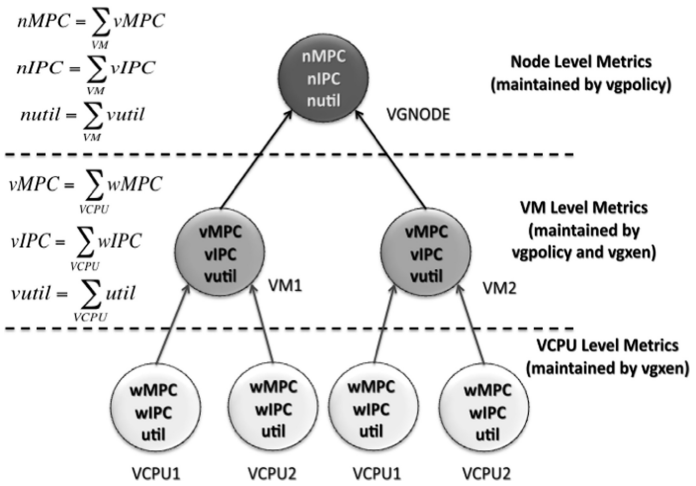


Fig. 3. An example of hierarchical metrics in vGreen.

and MPC) of running VMs. Since multiple VMs with possibly multiple VCPUs might be active concurrently, it is important to cleanly isolate the characteristics of each of these different entities. vGreen adopts a hierarchical approach for this purpose as illustrated in Figure 3. The lowest level of the hierarchy is the VCPU level, which is the fundamental unit of execution and scheduling in Xen. When a VCPU is scheduled on a PCPU by the Xen scheduler, *vgxen* starts the CPU performance counters of that PCPU to count the following events: *(1) Instructions Retired (INST)*, *(2) Clock cycles (CLK)*, and *(3) Memory accesses (MEM)*.

When that VCPU consumes its time slice (or blocks) and is removed from the PCPU, *vgxen* reads the performance counter values and estimates its MPC

(MEM/CLK) and IPC (INST/CLK) for the period it executed. This process is performed for every VCPU executing in the system across all the PCPUs. To effectively estimate the impact of these metrics on the VCPU power consumption and performance, *vgxen* also keeps track of the CPU utilization (*util*) of each VCPU, that is, how much time it actually spends executing on a PCPU over a period of time. This is important, since even a high IPC benchmark will cause high power consumption only if it is executing continuously on the PCPU. Hence, the metric derived for each VCPU is weighted by its *util*, and is referred to as the current weighted MPC and IPC ($wMPC_{cur}$ and $wIPC_{cur}$) as shown next.

$$wMPC_{cur} = MPC \cdot util$$
$$wIPC_{cur} = IPC \cdot util \quad (2)$$

They are referred to as "current", since they are estimated based on the IPC/MPC values from the latest run of a VCPU. To also take into account the previous value of these metrics, we maintain them as running exponential averages. The equation that follows shows how weighted MPC is estimated. We have

$$wMPC = \alpha \cdot wMPC_{cur} + (1 - \alpha) \cdot wMPC_{prev}, \quad (3)$$

where the new value of weighted MPC ($wMPC$) is calculated as an exponential average of $wMPC_{prev}$, the previous value of $wMPC$, and $wMPC_{cur}$ (Eq. (2)). The factor α determines the weight of current value ($wMPC_{cur}$) and history ($wMPC_{prev}$). In our implementation we use α=0.5, thus giving equal weight to both. The IPC metric is computed in a similar fashion as discussed before. We store these averaged metrics in the Xen VCPU structure to preserve them faithfully across VCPU context switches. This constitutes the metric estimation at the lowest level of the hierarchy as shown in Figure 3.

At the next level, *vgxen* estimates the aggregate metrics (vMPC, vIPC, vutil) for each VM by adding up the corresponding metrics of its constituent VCPUs, as shown in the middle level of Figure 3. This information is stored in VM structure of Xen to personalize metrics at per VM level and is exported to Dom-0 through a shared page, which is allocated by *vgxen* at the boot-up time.

vgdom. The second vGreen module of *vgnode* is the *vgdom* (see Figure 2). Its main role is to periodically (T_{up_period}) read the shared page exported by *vgxen* to get the latest characteristics metrics for all the VMs running on the vgnode, and update the *vgserv* with it. In addition, *vgdom* also acts as an interface for the *vgnode* to the *vgserv*. It is responsible for registering the *vgnode* with the *vgserv* and also for receiving and executing the commands sent by the *vgserv* as shown in Figure 2.

4.1.2 *vgserv*. The *vgserv* acts as the cluster controller and is responsible for managing VM scheduling and power management across the *vgnode* cluster. The *vgpolicy* is the core of *vgserv*, which makes the scheduling and power management decisions based on periodic updates on the VM metrics from the *vgnodes*. The metrics of each VM are aggregated by the *vgpolicy* to construct the top-level or node-level metrics (nMPC, nIPC, nutil) as shown in Figure 3.

Table I. MPC Balance Algorithm

Input: *vgnode* $n1$
1: if $nMPC_{n1} < nMPC_{th}$ **then**
2: **return**
3: **end if**
4: $pm\_min \leftarrow NULL$
5: **for** all vgnodes n_i except $n1$ **do**
6: **if** $(nMPC_{n_i} < nMPC_{th})$ and $(nMPC_{n1} - nMPC_{th}) < (nMPC_{th} - nMPC_{n_i})$ **then**
7: **if** $!pm\_min$ or $nMPC_{pm\_min} > nMPC_{n_i}$ **then**
8: $pm\_min \leftarrow n_i$
9: **end if**
10: **end if**
11: **end for**
12: $vm_{mig} \leftarrow NULL$
13: **for all** vm_i in $n1$ **do**
14: **if** $(nMPC_{th} - nMPC_{pm\_min}) > vMPC_{vm_i}$ and $vMPC_{vm_i} > vMPC_{vm_{mig}}$ **then**
15: $vMPC_{vm_{mig}} \leftarrow vMPC_{vm_i}$
16: **end if**
17: **end for**
18: **if** pm_min and vm_{mig} **then**
19: $do\_migrate(vm_{mig}, n1, pm\_min)$
20: **end if**

Thus, the knowledge of both the node-level and VM-level metrics allow the *vgpolicy* to understand not only the overall power and performance profile of the whole *vgnode*, but also fine-grained knowhow of the breakdown at VM level.

Based on these metrics, the *vgpolicy* runs its balancing and power management algorithm periodically (T_{p_period}). The basic algorithm is motivated by the fact that VMs with heterogeneous characteristics should be coscheduled on the same *vgnode* (Section 3). The problem of consolidation of VMs in minimum possible PMs has been explored in previous work [Hermenier et al. 2009; Wood et al. 2007], and is similar to bin-packing problem, which is computationally NP-hard. As discussed in Section 2, the existing solutions perform the consolidation based on just CPU utilization. Our balancing algorithms build on top of these existing algorithms to perform balancing based on MPC and IPC as well. The overall algorithm runs in the following four steps.

(1) *MPC balance*. This step ensures that nMPC is balanced across all the *vgnodes* in the system for better overall performance and energy efficiency across the cluster. Table I gives an overview of how the MPC balance algorithm works for a *vgnode* $n1$. The algorithm first of all checks if the nMPC of $n1$ is greater than a threshold $nMPC_{th}$ (step 1 in Table I). This threshold is representative of whether high MPC is affecting the performance of the VMs in that *vgnode*. This is based on the observation in Section 3, that for lower MPC workloads (like *eon*), the memory subsystem is lightly loaded and has little impact on the performance of the workload. Hence, if nMPC is smaller, the function returns, since there is no MPC balancing required for $n1$ (step 2 in Table I). If it is higher, then in steps 4–11, the algorithm tries to find the target *vgnode* with the minimum nMPC (pm_min) to which a VM from $n1$ could be migrated to resolve the MPC imbalance, subject to

the condition in step 6. The condition states that the target *vgnode* (n_i) nMPC ($nMPC_{n_i}$) must be below $nMPC_{th}$ by atleast ($nMPC_{n1} - nMPC_{th}$). This is required, since otherwise migration of a VM from $n1$ to n_i cannot bring $n1$ below the MPC threshold or might make n_i go above the MPC threshold. In steps 7 and 8, it stores the node n_i as target minimum nMPC *vgnode* (*pm_min*), if its nMPC ($nMPC_{n_i}$) is lower than the nMPC of the *vgnode* currently stored as *pm_min*. This way, once the loop in step 5 completes, it is able to locate the *vgnode* in the system with the least nMPC (*pm_min*).

Once the *pm_min* is found, the algorithm finds the VM (vm_{mig}) that could be migrated to *pm_min* for resolving the MPC imbalance (steps 12–17). For this purpose it scans the list of VMs on $n1$ to find the VM with the maximum vMPC, which if migrated, does not reverse the imbalance by making nMPC of *pm_min* more than $nMPC_{th}$ (steps 14 and 15). If such a VM is found, the algorithm invokes the *do_migrate* function to live migrate vm_{mig} from $n1$ to *pm_min* [Clark et al. 2005] in step 19. The decisions taken by the *vgpolicy* (updates, migration) are communicated to the *vgnodes* in form of commands as shown in Figure 2, while the *vgdom* component on the *vgnode* actually accomplishes the migration.

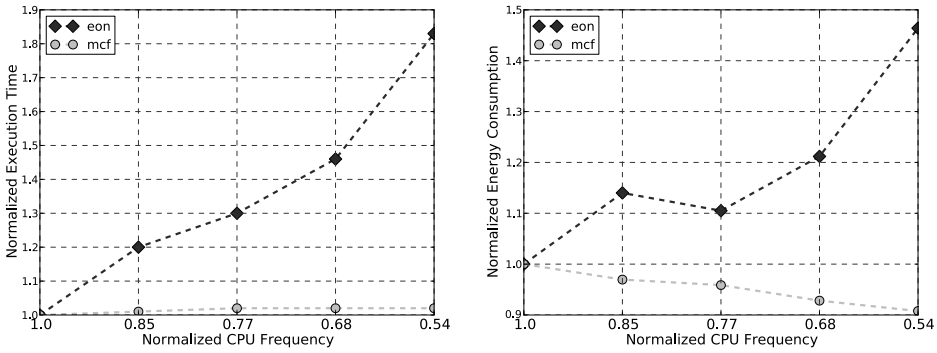
The complexity of the MPC balance algorithm (Table I) is linear ($O(n)$, where "n" is the number of *vgnodes* in steps 5–11, and number of VMs on $n1$ in steps 13–17) for resolving an MPC bottleneck, since it requires a single scan of vgnodes and VMs to detect and resolve it. Hence, in terms of implementation and performance the algorithm is simple and scalable.

(2) *IPC balance*. This step ensures nIPC is balanced across the *vgnodes* for better balance of power consumption across the PMs. The algorithm is similar to MPC balance, but uses nIPC instead of nMPC.

(3) *Util balance*. This step balances the CPU utilization of *vgnodes* to ensure there are no overcommitted nodes in the system, if there are other underutilized *vgnodes*. The algorithm is again similar to MPC balance, but uses nutil instead of nMPC.

(4) *Dynamic Voltage Frequency Scaling (DVFS)*. The *vgpolicy* may issue a command to scale the voltage-frequency setting (v-f setting) of a *vgnode*, if it deems that it is more energy efficient than VM migration. This may happen if there are not enough heterogeneous VMs across the cluster to be able to balance the resource utilization evenly. The DVFS policy is itself based on state-of-the-art DVFS policies [Dhiman and Rosing 2007; Isci et al. 2006] that exploit the characteristics of the workload to determine the best suited v-f setting for it. Specifically, it aggressively downscales the v-f setting if the overall MPC is high ($> nMPC_{th}$), otherwise keeps the system at the highest v-f setting.

Figure 4 gives the intuition behind the policy using an example of two benchmarks, *mcf* and *eon*, running at 90% CPU utilization level. It plots the execution time (Figure 4(a)) and energy consumption (Figure 4(b)) at five different v-f settings. The execution time, energy consumption, and the v-f settings are normalized against the values at the highest v-f setting. We can observe that as the frequency is decreased, the execution time of *eon* almost increases in



(a) normalized execution time (b) normalized energy consumption

Fig. 4. Comparison of execution time and energy consumption of *mcf* and *eon* at different frequency levels

proportion to the drop in frequency. For instance, at normalized frequency of 0.54, the increase in execution time is more than 80% ($\sim \frac{1}{0.54}$). This happens since *eon* has high IPC, and uses the pipeline of the processor intensively, which makes its execution time a function of the clock rate of the pipeline or the CPU frequency. This huge performance degradation has a direct impact on the energy consumption of eon at lower v-f settings as shown in Figure 4(b). We can observe that at all the frequencies the system consumes more energy compared to the highest v-f setting, reaching as high as 40% more. This implies, that for high IPC workloads, DVFS is actually energy inefficient.

In contrast, for *mcf*, which has high MPC, we observe that the execution time (Figure 4(a)) is actually fairly independent of the CPU frequency. This is a consequence of the high degree of CPU stalls that occur during its execution due to frequent memory accesses, which makes its execution time insensitive to actual CPU frequency. The low performance degradation translates into system-level energy savings (see Figure 4(b)), which reaches 10% at the lowest frequency.

This example also illustrates the fact that the effectiveness of DVFS for energy savings is not very significant in modern server class systems. This has been also observed in previous research [Dhiman et al. 2008; Meisner et al. 2009], and the reasons for such a trend include lower contribution of CPU in total system power consumption, finer voltage settings in modern CPUs due to shrinking process technology, etc. These observations also motivate our approach to focus more on efficient VM scheduling to achieve higher energy savings rather than on aggressive DVFS. Rather, the system resorts to DVFS only when no further benefits are achievable through scheduling and the MPC is high enough to achieve energy savings. As we show in Section 5, such an approach enables energy savings under both heterogeneous and homogeneous workload scenarios through VM scheduling and aggressive DVFS, respectively.

The four steps described earlier in the overall algorithm have relative priorities to resolve conflicts, if they occur. MPC balance is given the highest priority, since a memory bottleneck severely impacts overall performance and energy

efficiency as identified in Section 3.1 (Figure 1(a)). IPC balance results in a more balanced power consumption profile, which helps create an even thermal profile across the cluster and hence reduces cooling costs [Ayoub et al 2010], and is next in the priority order. Finally, Utilization balance results in a fairly loaded system, and is representative of the prior state-of-the-art scheduling algorithms. DVFS step (step 4), as explained before, is invoked only if the system is already balanced from the perspective of MPC, IPC, and CPU utilization, and no further savings are possible through VM scheduling.

4.2 Implementation

Our testbed for vGreen includes two state-of-the-art 45nm Dual Intel Quad Core Xeon X5570 (Intel Nehalem architecture with 16 PCPUs each)-based server machines with 24GB of memory, which act as the *vgnodes*, and a Core2Duo-based desktop machine that acts as the *vgserv*. The *vgnodes* run Xen3.3.1, and use Linux 2.6.30 for Dom-0.

The *vgxen* module is implemented as part of the Xen credit scheduler (the default scheduler) to record VCPU- and VM- level metrics. It stores all the VM-level metrics in a shared page mapped into the address space of Dom-0. It further exposes a new hypercall which allows the *vgdom* to map this shared page into its address space when it gets initialized (as explained in Section 4.1.1). The *vgdom* module is implemented in two parts on Dom-0.

(1) *vgdom Driver*. A Linux driver that interfaces with *vgxen* to get the VM characteristics and exposes it to the application layer. When initialized, it maps the shared page storing the VM metrics into its address space using the hypercall discussed earlier. Such a design makes getting the VM metrics a very low overhead process, since it is a simple memory read.
(2) *vgdom App*. An application client module that is responsible for interfacing and registering the *vgnode* with the *vgserv*. The primary responsibility is to get the VM metrics data from the driver and pass it on to *vgserv* as shown in Figure 2. It also accepts *vgserv* commands for VM migration or DVFS and processes it.

vGreen requires no modifications to either the OS or the application running within the VMs. This makes the system nonintrusive to customer VMs and applications and hence easily deployable on existing clusters. The *vgserv* and *vgpolicy* run on Linux 2.6.30, and are implemented as application server modules.

The system is designed to seamlessly handle dynamic entry and exit of *vgnodes* in the system without any disruption to the *vgpolicy*. On initialization, *vgserv* opens a well-known port and waits for new *vgnodes* to register with it. When *vgnodes* connect, *vgserv* instructs them to regularly update it with VM characteristics (T_{up_period}), and accordingly updates the node- and VM-level metrics. It runs the *vgpolicy* every T_{p_period} and performs balancing or DVFS decisions, which it communicates to the *vgnode* through commands as described in Section 4.1. If a *vgnode* goes offline, the connection between it and

Table II. Benchmarks Used

Benchmark	Characteristics
eon	High IPC/Low MPC
applu	Medium IPC/High MPC
perl	High IPC/Low MPC
bzip2	Medium IPC/Low MPC
equake	Low IPC/High MPC
gcc	High IPC/Low MPC
swim	Low IPC/High MPC
mesa	High IPC/Low MPC
art	Medium IPC/High MPC
mcf	Low IPC/High MPC

the *vgserv* is broken. This results in a dynamic cleanup of all the state associated with that *vgnode* on the *vgserv*.

For real-world deployments, vGreen can be either installed as a stand-alone system for VM management, or as part of bigger infrastructure management systems like OpenNebula [OpenNebula] or Grid Virtualization Engine [Wang et al. 2009] as well. For instance, in context of GVE, *vgxen* and *vgdom* can be incorporated into the "*GVE agent service layer*", which is the monitoring layer, while *vgserv* and *vgpolicy* can be implemented as part of the "*GVE site service layer*", which is the control layer.

5. EVALUATION

5.1 Methodology

For our experiments, we use benchmarks with varying characteristics from the SPEC-CPU 2000 benchmark suite. The used benchmarks and their characteristics are illustrated in Table II. We run each of these benchmarks inside a VM, which is initialized with eight VCPUs and 4GB of memory. We generate experimental workloads by running multiple VMs together, each running one of the benchmarks. For each combination run we sample the system power consumption of both the vgnodes every 2s using the power sensors in the PM, which we query through the IPMI interface [IPMI 2004].

We compare vGreen to a VM scheduler that mimics the Eucalyptus VM scheduler [Nurmi et al. 2008] for our evaluation. Eucalyptus is an open-source cloud computing system that can manage VM creation and allocation across a cluster of PMs. The default Eucalyptus VM scheduler assigns VMs using a greedy policy, that is, it allocates VMs to a PM until its resources (number of CPUs and memory) are full. However, this assignment is static, and it does not perform any dynamic VM migration based on actual PM utilization at runtime. For fair comparison, we augment the Eucalyptus scheduler with the CPU utilization metrics and algorithm proposed in the previous section, which allow it to redistribute/consolidate VMs dynamically at runtime. This enhancement is representative of the metrics employed by the existing state-of-the-art policies which use CPU utilization for balancing (see Section 2). We refer to this enhanced scheduler as E+. For further fairness in comparison, we use the same

initial assignment of VMs to PMs as done by the default Eucalyptus scheduler for both E+ and vGreen.

We report the comparative results of vGreen and E+ for two primary parameters:

(1) *System Level Energy savings*. We estimate the energy reduction in executing each combination of VMs using vGreen over E+. This is calculated by measuring the total system level energy consumption for a VM combination with E+ and vGreen, and then taking their difference. Note that the combinations may execute for different times with E+ and vGreen, and since we do not know the state of the system after the execution (could be active if there are more jobs, or be in sleep state if nothing to do), we only compare the energy consumed during active execution of each combination.

(2) *Average Weighted Speedup*. We also estimate the average speedup of each VM combination with vGreen. For this, we use the weighted speedup (AWS) based on a similar metric defined earlier in Section 3 (refer to Eq. (1)). It is defined as

$$AWS = \frac{\sum_{VM_i} \frac{T_{e+_i}}{T_{alone_i}}}{\sum_{VM_i} \frac{T_{vgreen_i}}{T_{alone_i}}} - 1, \tag{4}$$

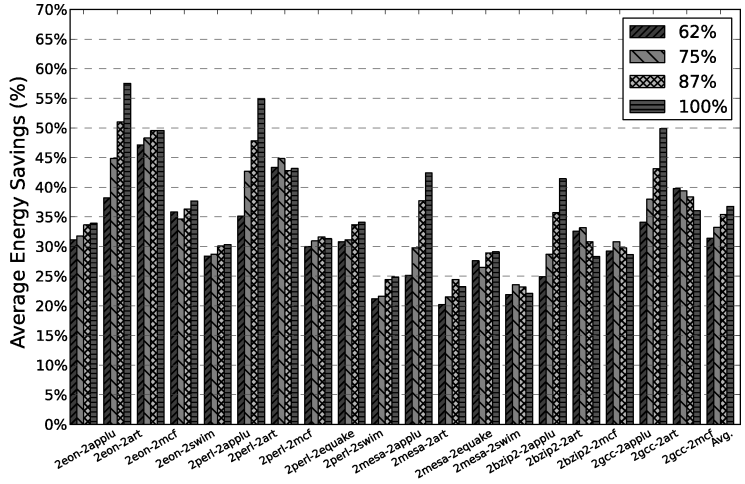
where T_{alone_i} is the execution time of VM_i when it runs alone on a PM, and T_{e+_i} and T_{vgreen_i} are its execution time as part of a VM combination with E+ and vGreen, respectively. AWS > 0 implies that the VM combination runs faster with vGreen and vice versa.

For all our experiments, we use P_{p_period} and P_{up_period} as 5s. Based on our experiments across different benchmarks, we choose $nMPC_{th}$ as 0.02 and $nIPC_{th}$ as 8. These threshold values allowed us to cleanly separate memory- and CPU-intensive VMs from each other.

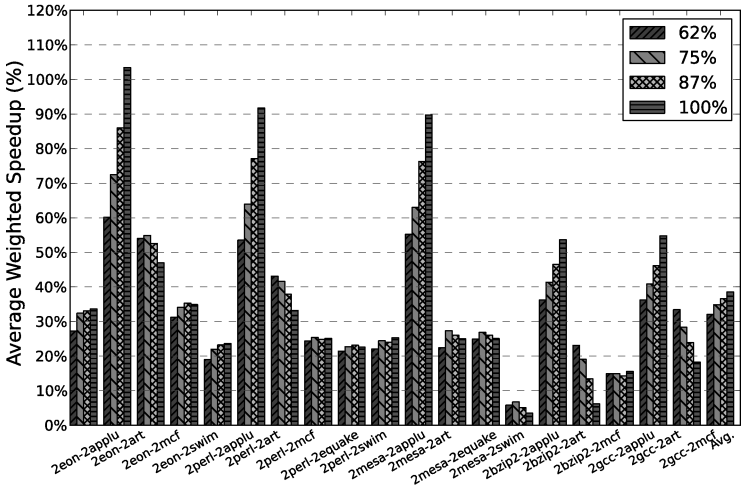
5.2 Results

5.2.1 *Heterogeneous Workloads.* In the first set of experiments, we use combinations of VMs running benchmarks with heterogeneous characteristics. Each VM consists of multiple instances of the benchmark to generate different CPU utilization levels. In total we run four VMs, varying the overall CPU utilization of *vgnodes* between 50% to 100%. We choose this range of CPU utilization, since it is representative of a consolidated environment, where multiple VMs are consolidated to get higher overall resource utilization across the cluster [Wood et al. 2007]. We run CPU-intensive benchmarks in two VMs, and memory-intensive benchmarks in the other two. We did experiments across all possible heterogeneous VM combinations, but for the sake of clarity and brevity have included results for 19 workloads in the following discussion. The excluded results lead to similar average metrics and conclusions as reported next.

Overall Results. Figure 5 shows the overall results across different utilization levels for the vGreen system normalized to that with E+. The x-axis on



(a) system-level energy savings



(b) average weighted speedup

Fig. 5. Comparison of E+ and vGreen.

the graphs shows the initial distribution of VMs on the physical machines by the default Eucalyptus scheduler. For instance, *2gcc/2art* means that two VMs running *gcc* are on the first PM, while the two VMs running *art* are on the second. We can observe in Figure 5(a), that vGreen achieves an average of between 30–40% system-level energy savings across all the utilization levels, reaching as high as 60%. The high energy savings are a result of the fact that vGreen schedules the VMs in a much more efficient fashion resulting in higher speedups while maintaining similar average power consumption. This results in energy savings, since now the benchmarks run and consume active power for a smaller duration.

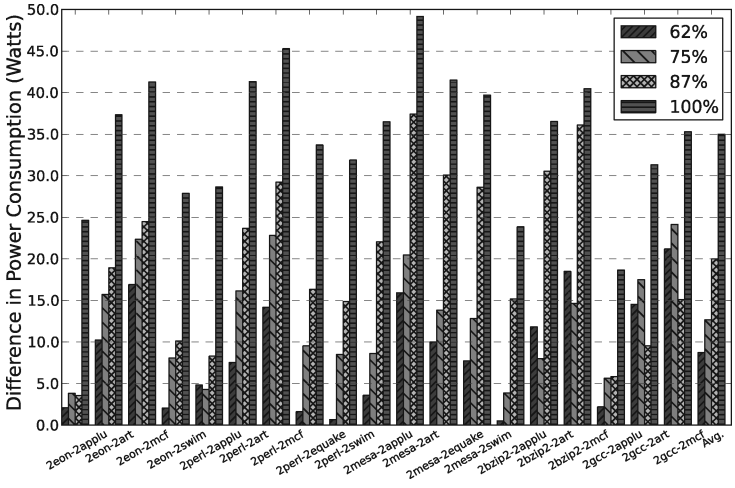


Fig. 6. Power consumption imbalance in E+: The difference in power consumption between the two PMs under the E+ scheduling algorithm.

Figure 5(b) shows that vGreen achieves an average of around 30–40% weighted speedup over E+ across all the combinations at all utilization levels, reaching as high as 100%. The reason for this is that E+ colocates the high-IPC VMs on one *vgnode*, and the high-MPC ones on the second one. Thereafter, since the CPU utilization of both the *vgnodes* is balanced, no dynamic relocation of VMs is done. With vGreen, although the initial assignment of the VMs is same as with E+, the dynamic characterization of VMs allows the *vgserv* to detect a heavy MPC imbalance. This initiates migration of a high-MPC VM to the second *vgnode* running the high-IPC VMs. This results in an IPC and utilization imbalance between the two *vgnodes*, since the second *vgnode* now runs a total of three high-utilization VMs. This is detected by *vgserv* and it responds by migrating a high-IPC VM to the first *vgnode*. This creates a perfect balance in terms of MPC, IPC, and utilization across both the *vgnodes*. This results in significant speedup as observed in Figure 5(b). We can see in Figure 5(b), that some combinations achieve higher weighted speedup compared to others. For instance, for the *2eon/2applu* combination it is around 30%, while for *2eon/2art* it is over 100%. This difference is due to the fact that colocation of *art* and *eon* VMs significantly benefits *art* from the point of view of larger cache and memory bandwidth availability, since it has very high MPC. In contrast, *applu* benefits lesser due to its lower overall MPC compared to *art*, which results in relatively smaller weighted speedup.

Another disadvantage of not taking the characteristics of the workload into account for scheduling is that there could be significant imbalance in power consumption across the nodes in a cluster. For instance, the node running high-IPC workloads might have much higher power consumption compared to the node running high-MPC workloads (as observed in Section 3). This can create power hot spots on certain nodes in the cluster, and be detrimental to the overall cooling energy costs [Ayoub et al 2010]. Figure 6 illustrates the

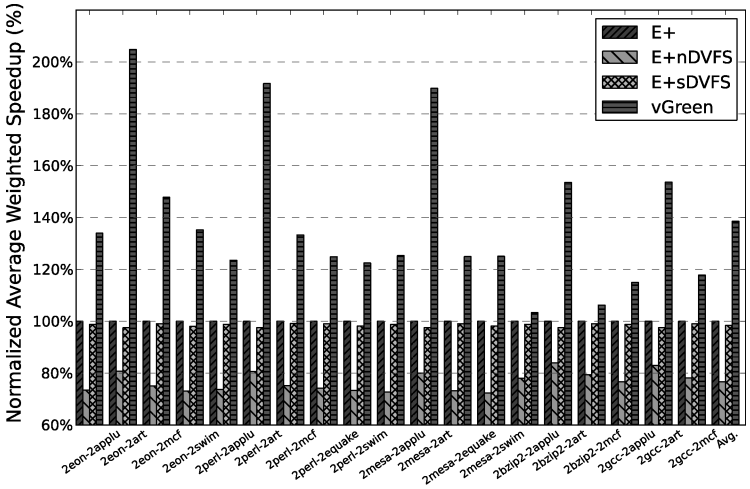
imbalance in power consumption across the two *vgnodes* under the E+ system. We can see that the average imbalance in power consumption could be as high as 30W, with the highest imbalance close to 45W. With vGreen system, this imbalance is almost negligible due to the better balance of IPC and utilization across the machines. This results in a better overall thermal and power profile and reduces power hot spots in the cluster.

Comparison with DVFS policies. A possible way for saving energy with the E+ system is to augment it with a DVFS policy. For comparison, we consider two policies for the E+ system.

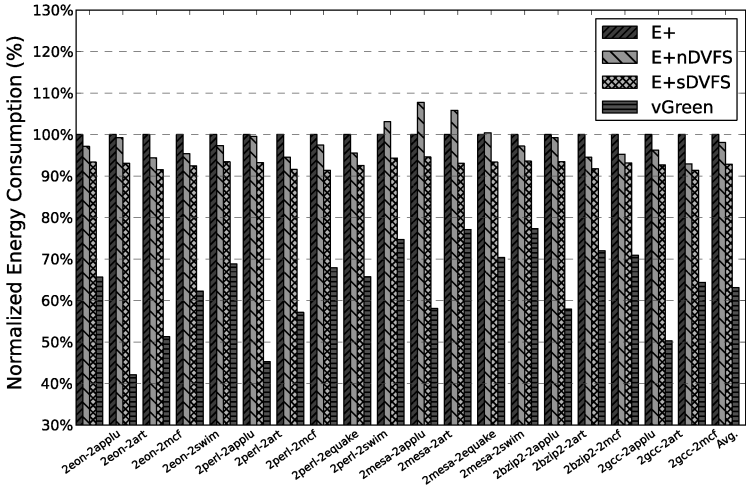
(1) The "naive" policy. This policy simply resorts to throttling the CPU in order to reduce the energy consumption in the system. We refer to the system with the "naive" policy as E+nDVFS.
(2) The "smart" policy. This policy is the same as incorporated into the vGreen system (see Section 4). The policy throttles the CPU only if it deems it would result in lower performance impact and higher energy savings. We refer to the system with the "smart" policy as E+sDVFS.

Figure 7 shows the average weighted speedup and energy consumption results for the E+sDVFS, E+nDVFS, and the vGreen system normalized against the results for the E+ system. Figure 7(a) illustrates the average weighted speedup results across all the combinations at 100% CPU utilization. The vGreen results are the same as those plotted in Figure 5(b), but have been included for the sake of comparison. We can observe that across all the combinations, both the DVFS policies perform slower than the baseline E+ system. This is intuitive, since the DVFS policies run the system at a lower frequency. However, the E+sDVFS clearly outperforms the E+nDVFS system across all workload combinations. While the E+sDVFS system is on an average always within 2% of the E+ system, the E+nDVFS system is on average 22% slower than the E+ system. This happens, since the E+sDVFS system exploits the characteristics of the VMs, and performs aggressive throttling only on the nodes running VMs with high MPC. As discussed in the Section 4, this results in minimal performance degradation, since such high-MPC workloads are highly stall intensive and have little dependence on CPU frequency. In contrast, the E+nDVFS system naively throttles even the nodes running high-IPC VMs, resulting in the high performance slowdown as observed in Figure 7(a).

The average weighted speedups have a direct impact on the energy savings as illustrated in Figure 7(b). The E+nDVFS system gets an average of just 1% energy savings across all the combinations. For some workloads, like *2mesa/2art*, it in fact consumes more energy than the baseline system. This indicates that the power reduction due to E+nDVFS system is outweighed by the huge performance slowdown. The E+sDVFS system does better by achieving around 9% energy savings due to the small performance slowdown. However, both are clearly outperformed by the vGreen system, which achieves close to 35% energy savings. This shows that efficient resource utilization across a cluster is key to energy-efficient computing in virtualized environments.



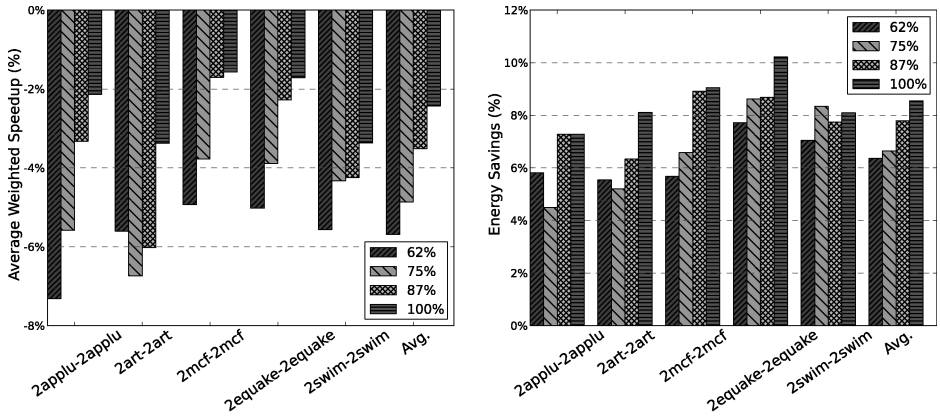
(a) normalized average weighted speedup



(b) normalized energy consumption

Fig. 7. Comparison of E+, E+nDVFS, E+sDVFS, and vGreen.

5.2.2 *Homogeneous Workloads.* We also experimented with combination of VMs running homogeneous benchmarks to evaluate the performance of our system under cases where there is no heterogeneity across VMs. We did experiments for all the benchmarks in Table II, where all the four VMs ran the same benchmark. We observed that in all the experiments, there was no possibility of rebalancing based on characteristics, since the MPC and IPC of the VMs were already balanced. However, for the case of high-MPC workloads, the vGreen system effectively applies DVFS to get energy savings. Figure 8 illustrates the average weighted speedup and energy savings achieved across the homogeneous set of high-MPC workloads. We can observe that vGreen achieves



(a) average weighted speedup (b) system-level energy savings

Fig. 8. Comparison of E+ and vGreen with homogeneous workloads.

Table III. Comparison of Machines

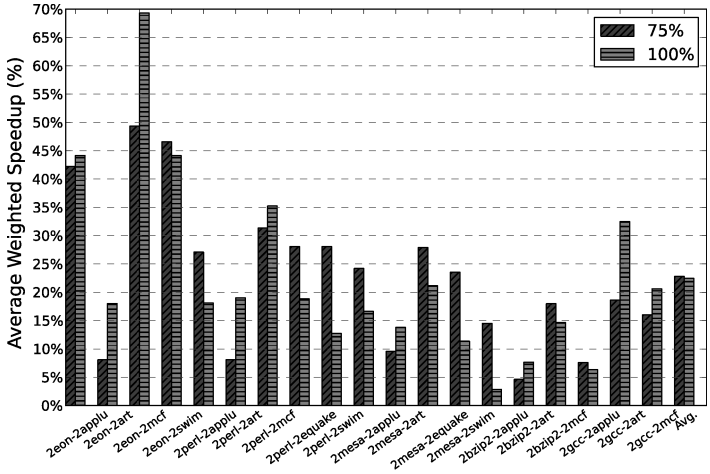
Characteristic	Machine-1	Machine-2
Microarchitecture	Intel Nehalem	Intel Core
CPU	Xeon X5570	Xeon E5440
# of PCPUs	16	8
Caches	L1-L2-L3	L1-L2
Thermal Design Power	95W	80W
Memory	24GB	8GB
Memory Type	DDR3	DDR2
Memory Controller	On-Die (2.93GHz)	Off-Chip (1.33GHz)
Memory Channels	3	2

average system-level energy savings of between 6–9% across all the utilization levels. The slowdown due to DVFS is between 2–5% as indicated in Figure 8(a). For high-IPC workloads, the results were identical to E+ system, since vGreen neither does any VM migration nor DVFS.

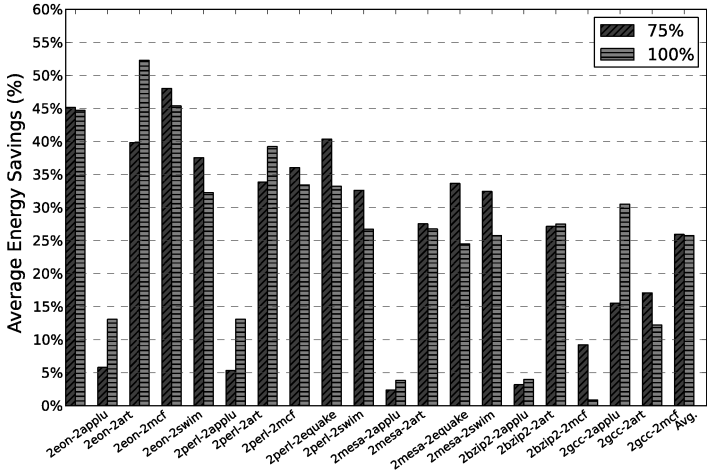
5.3 Different Machine Architecture and Configurations

To verify the scalability of our system and ideas, we also experimented on a machine with a different microarchitecture and configuration. Table III compares the configurations of the two machines: Machine-1 refers to the Intel Nehalem-based machine, which we used in the earlier parts of the evaluation section; Machine-2 refers to the other machine, for which we present results in this section. A quick look at Table III shows that the two machines significantly vary from each other in terms of CPU microarchitecture (Intel Nehalem versus Intel Core), power characteristics (different TDP), caches (L3 versus L2), as well as the memory technology (DDR3 versus DDR2 and on-die versus off-chip memory controller).

The methodology for experiments on Machine-2 was the same as that for Machine-1 with small changes: (1) We use 2GB as the memory size for each VM due to less memory in Machine-2; (2) we do experiments for 75% and 100%



(a) average weighted speedup



(b) system-level energy savings

Fig. 9. Comparison of E+ and vGreen on the Intel Core-based machine.

utilization only, since the machine has 8 PCPUs. The 62% and 87% workloads on this machine would need a total of 5 and 7 threads, respectively, which cannot be evenly divided across two VMs.

Figure 9 shows the average weighted speedup and energy savings results across the heterogeneous set of workloads. We can observe that vGreen achieves close to 22% average weighted speedup across both the utilization levels. Similar to Figure 5, we can observe that for some workloads (e.g., 70% for *2eon/2mcf*), the speedup is more than others (e.g., 45% for *2eon/2swim*). This is again due to the fact that some workloads benefit more due to the efficient resource utilization than others due to their higher aggregate MPC (*mcf*

having higher MPC than *swim* in this case). This speedup results in around 25% system-level energy savings across both the utilization levels.

A quick comparison of overall results with Figure 5 indicates that both the average weighted speedup and energy savings are higher on the Machine-1 by around 10%. This is explained by the faster memory technology (DDR3), memory controller (on-die), and higher number of channels (3) of Machine-1 compared to Machine-2 (refer to Table III). As a consequence of a faster memory subsystem, the high-MPC workloads benefit more on Machine-1 than Machine-2, when the memory subsystem load is relieved, that is, they run much faster on Machine-1. Since vGreen balances the aggregate MPC of workloads across the machines, it results in higher weighted speedup on Machine-1 compared to Machine-2. In future systems, the machine architectures are going to move towards even faster memory technology and architectures, and these results indicate that a vGreen-like system is even more beneficial for exploiting their design, and delivering higher performance and energy efficiency.

5.4 Overhead

In our experiments we observed negligible runtime overhead due to vGreen. On the *vgnodes*, *vgxen* is implemented as a small module which does simple performance counter operations and VCPU and VM metric updates. The performance counters are hardware entities with negligible cost (order of 100 cycles) on software execution as accessing them is just a simple register read/write operation. The *vgdom* executes every T_{up_period} (5s in our experiments), and as explained in Section 4 just reads and transmits the VM metrics information to the *vgserv*. In our experiments, we observed negligible difference in execution time of all the benchmarks ($< 1\%$) with and without *vgxen* and *vgdom*.

vGreen achieves energy efficiency through VM scheduling, which requires VM migration. We observed negligible overhead of VM live migration on execution times of benchmarks, which is consistent with the findings in [Clark et al. 2005]. VM migration, however, involves extra activity in the system on both the source and the destination PMs. The primary source of activity is the processing of network packets of VM data, and processing associated with creating new VM on the destination and cleanup on the source. However, our methodology takes all of these costs into account. As described in Section 5.1, we record the performance of the benchmarks within the VMs and sample power consumption of the whole server in a power log every 2s for the entire run. If a VM migration occurs in between, the extra power consumption due to VM migration-related processing (discussed before) on the network card, Dom0 and the hypervisor is taken in to account in the power log. The impact of extra processing due to VM migration on the performance of the benchmark is also taken in to account, since we record the execution times of the benchmarks. These recorded power and performance numbers are used to estimate the energy savings and average weighted speedup (Eq. (4)) for vGreen. Hence, all our results in Section 5 already incorporate these power and performance overheads, which indicate that the cost is clearly overwhelmed by the benefits of VM migration.

6. CONCLUSION

In this article we presented vGreen, a system for energy-efficient VM management across a cluster of machines. The key idea behind vGreen is linking workload characterization of VMs to VM scheduling and power management decisions to achieve better performance, energy efficiency, and power balance in the system. We designed novel hierarchical metrics to capture VM characteristics, and develop scheduling and DVFS policies to achieve the aforementioned benefits. We implemented vGreen on a real-life testbed of state-of-the-art server machines, and show with benchmarks with varying characteristics that it can achieve improvement in average performance and system-level energy savings of up to 40% over state-of-the-art policies at a very low overhead. We further demonstrate the applicability and scalability of the system across machines with different architecture and configurations.

REFERENCES

ABDELSALAM, H. S., MALY, K., MUKKAMALA, R., ZUBAIR, M., AND KAMINSKY, D. 2009. Analysis of energy efficiency in clouds. In *Future Computing, Service Computation, Cognitive, Adaptive, Content, Patterns, Computation World*, 416–421.

AMAZON. 2008. *Amazon Elastic Compute Cloud (Amazon EC2)*. http://aws.amazon.com/ec2/.

AYOUB, R., SHERIFI, S., AND ROSING, T. 2010. Gentlecool: Cooling aware proactive workload scheduling in multi-machine systems. In *Proceedings of the IEEE Design, Automation Test in Europe (DATE'10)*.

BARHAM, P., DRAGOVIC, B., FRASER, K., HAND, S., HARRIS, T., HO, A., NEUGEBAUER, R., PRATT, I., AND WARFIELD, A. 2003. Xen and the art of virtualization. In *Proceedings of the 19th ACM Symposium on Operating Systems Principles (SOSP'03)*. ACM, New York, 164–177.

BOBROFF, N., KOCHUT, A., AND BEATY, K. 2007. Dynamic placement of virtual machines for managing sla violations. In *Integrated Network Management*. IEEE, 119–128.

CHASE, J. S., ANDERSON, D. C., THAKAR, P. N., VAHDAT, A. M., AND DOYLE, R. P. 2001. Managing energy and server resources in hosting centers. In *Proceedings of the 18th ACM Symposium on Operating Systems Principles (SOSP'01)*. ACM, New York, 103–116.

CLARK, C., FRASER, K., HAND, S., HANSEN, J. G., JUL, E., LIMPACH, C., PRATT, I., AND WARFIELD, A. 2005. Live migration of virtual machines. In *Proceedings of the 2nd Conference on Symposium on Networked Systems Design & Implementation (NSDI'05)*. USENIX Association, 273–286.

DHIMAN, G., MARCHETTI, G., AND ROSING, T. 2009. vGreen: A system for energy efficient computing in virtualized environments. In *Proceedings of the International Symposium on Lower Power Electronics and Design (ISLPED'09)*. ACM, New York.

DHIMAN, G., MIHIC, K., AND ROSING, T. 2010. A system for online power prediction in virtualized environments using gaussian mixture models. In *Proceedings of the 47th Design Automation Conference (DAC'10)*. ACM, New York, 807–812.

DHIMAN, G., PUSUKURI, K., AND ROSING, T. S. 2008. Analysis of dynamic voltage scaling for system level energy management. In *Proceedings of the Workshop on Power Aware Computing and Systems (HotPower'08)*.

DHIMAN, G. AND ROSING, T. S. 2007. Dynamic voltage frequency scaling for multi-tasking systems using online learning. In *Proceedings of the International Symposium on Low Power Electronics and Design (ISLPED'07)*. ACM, New York, 207–212.

FAN, X., WEBER, W.-D., AND BARROSO, L. A. 2007. Power provisioning for a warehouse-sized computer. In *Proceedings of the 34th Annual International Symposium on Computer Architecture (ISCA'07)*. ACM, New York, 13–23.

GE, R., FENG, X., FENG, W.-C., AND CAMERON, K. W. 2007. Cpu miser: A performance-directed, run-time system for power-aware clusters. In *Proceedings of the International Conference on Parallel Processing (ICPP'07)*. IEEE Computer Society, 18.

HALETKY, E. L. 2008. *VMware ESX Server in the Enterprise: Planning and Securing Virtualization Servers*. Prentice Hall.

HERMENIER, F., LORCA, X., MENAUD, J.-M., MULLER, G., AND LAWALL, J. 2009. Entropy: a consolidation manager for clusters. In *Proceedings of the ACM SIGPLAN/SIGOPS International Conference on Virtual Execution Environments (VEE'09)*. ACM, New York, 41–50.

IPMI. 2004. Intelligent platform management interface v2.0 specification. http://www.intel.com/design/servers/impi.

ISCI, C., CONTRERAS, G., AND MARTONOSI, M. 2006. Live, runtime phase monitoring and prediction on real systems with application to dynamic power management. In *Proceedings of the 39th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO'39)*. IEEE Computer Society, 359–370.

KANSAL, A., ZHAO, F., LIU, J., KOTHARI, N., AND BHATTACHARYA, A. A. 2010. Virtual machine power metering and provisioning. In *Proceedings of the 1st ACM Symposium on Cloud Computing (SoCC'10)*. ACM, New York, 39–50.

KNAUERHASE, R. C., BRETT, P., HOHLT, B., LI, T., AND HAHN, S. 2008. Using os observations to improve performance in multicore systems. *IEEE Micro 28*, 3, 54–66.

KOLLER, R., VERMA, A., AND NEOGI, A. 2010. Wattapp: An application aware power meter for shared data centers. In *Proceeding of the 7th International Conference on Autonomic Computing (ICAC'10)*. ACM, New York, 31–40.

LIU, L., WANG, H., LIU, X., JIN, X., HE, W. B., WANG, Q. B., AND CHEN, Y. 2009. Greencloud: a new architecture for green data center. In *Proceedings of the 6th International Conference Industry Session on Autonomic Computing and Communications Industry Session (ICAC-INDST'09)*. ACM, New York, 29–38.

MCNETT, M., GUPTA, D., VAHDAT, A., AND VOELKER, G. M. 2007. Usher: An extensible framework for managing custers of virtual machines. In *Proceedings of the 21st Conference on Large Installation System Administration Conference (LISA'07)*. USENIX Association, 1–15.

MEISNER, D., GOLD, B., AND THOMAS, W. 2009. Powernap: Eliminating server idle power. In *Proceedings of the 14th International Conference on Architectural Support for Programming Languages and Operating Systems*.

MERKEL, A. AND BELLOSA, F. 2006. Balancing power consumption in multiprocessor systems. *SIGOPS Oper. Syst. Rev. 40*, 4, 403–414.

MERKEL, A., STOESS, J., AND BELLOSA, F. 2010. Resource-conscious scheduling for energy efficiency on multicore processors. In *Proceedings of the 5th European Conference on Computer Systems (EuroSys'10)*. ACM, New York, 153–166.

MOORE, J., CHASE, J., RANGANATHAN, P., AND SHARMA, R. 2005. Making scheduling "cool": Temperature-aware workload placement in data centers. In *Proceedings of the Annual Conference on USENIX Annual Technical Conference (ATEC'05)*. USENIX Association, 5–5.

NATHUJI, R., ENGLAND, P., SHARMA, P., AND SINGH, A. 2009. Feedback driven qos-aware power budgeting for virtualized servers. In *Proceedings of the 4th International Workshop on Feedback Control Implementation and Design in Computing Systems and Networks (FeBID'09)*.

NATHUJI, R., KANSAL, A., AND GHAFFARKHAH, A. 2010. Q-clouds: Managing performance interference effects for qos-aware clouds. In *Proceedings of the 5th European Conference on Computer Systems (EuroSys'10)*. ACM, New York, 237–250.

NATHUJI, R. AND SCHWAN, K. 2007. Virtualpower: Coordinated power management in virtualized enterprise systems. In *Proceedings of 21st ACM SIGOPS Symposium on Operating Systems Principles (SOSP'07)*. ACM, New York, 265–278.

NURMI, D., WOLSKI, R., GRZEGORCZYK, C., OBERTELLI, G., SOMAN, S., YOUSEFF, L., AND ZAGORODNOV, D. 2008. The eucalyptus open-source cloud-computing system. In *Proceedings of Cloud Computing and Its Applications*.

OPENNEBULA. Opennebula homepage. http://dev.opennebula.org/

PAKBAZNIA, E. AND PEDRAM, M. 2009. Minimizing data center cooling and server power costs. In *Proceedings of the International Symposium on Lower Power Electronics and Design (ISLPED'09)*. ACM, 145–150.

RAGHAVENDRA, R., RANGANATHAN, P., TALWAR, V., WANG, Z., AND ZHU, X. 2008. No "power" struggles: Coordinated multi-level power management for the data center. In *Proceedings of the 13th International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS'08)*. ACM, New York, 48–59.

RANGANATHAN, P., LEECH, P., IRWIN, D., AND CHASE, J. 2006. Ensemble-level power management for dense blade servers. In *Proceedings of the 33rd Annual International Symposium on Computer Architecture (ISCA'06)*. IEEE Computer Society, 66–77.

SNAVELY, A. AND TULLSEN, D. M. 2000. Symbiotic jobscheduling for a simultaneous mutlithreading processor. *SIGPLAN Not. 35*, 11, 234–244.

STOESS, J., LANG, C., AND BELLOSA, F. 2007. Energy management for hypervisor-based virtual machines. In *Proceedings of the USENIX Annual Technical Conference (ATC'07)*. USENIX Association, 1–14.

VERMA, A., AHUJA, P., AND NEOGI, A. 2008. Power-Aware dynamic placement of hpc applications. In *Proceedings of the 22nd Annual International Conference on supercomputing (ICS'08)*. ACM, New York, 175–184.

VMWARE. 2009. Vmware distributed resource scheduler. http://www.vmware.com/products/drs/

WANG, L., VON LASZEWSKI, G., TAO, J., AND KUNZE, M. 2009. Grid virtualization engine: design, implementation and evaluation. *IEEE Syst. J. 3*, 4, 477–488.

WANG, R. AND KANDASAMY, N. 2009. A distributed control framework for performance management of virtualized computing environments: Some preliminary results. In *Proceedings of the 1st Workshop on Automated Control for Datacenters and Clouds (ACDC'09)*. ACM, New York, 7–12.

WOOD, T., SHENOY, P., AND ARUN. 2007. Black-Box and gray-box strategies for virtual machine migration. In *Proceedings of the ACM Symposium on Networked Systems Design and Implementation (NSDI'07)*. 229–242.

Received March 2010; revised September 2010; accepted September 2010