vGreen: A System for Energy Efficient Computing in Virtualized Environments *

Gaurav Dhiman
gdhiman@cs.ucsd.edu

Giacomo Marchetti
gmarchet@ucsd.edu

Tajana Rosing
tajana@ucsd.edu

Department of Computer Science and Engineering
University of California, San Diego
La Jolla, CA 92093-0404

ABSTRACT

In this paper, we present vGreen, a multi-tiered software system for energy efficient computing in virtualized environments. It comprises of novel hierarchical metrics that capture power and performance characteristics of virtual and physical machines, and policies, which use it for energy efficient virtual machine scheduling across the whole deployment. We show through real life implementation on a state of the art testbed of server machines that vGreen can improve both performance and system level energy savings by 20% and 15% across benchmarks with varying characteristics.

Categories and Subject Descriptors

D.4.7 [**Operating Systems**]: Distributed Systems

General Terms

Design, Experimentation, Performance

Keywords

Virtualization, Migration, Energy, Workload Characterization

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 is a major component of these costs. For instance, for a 10MW data center, this can range in the order of millions of dollars per year [15].

Modern data centers use virtualization (eg. Xen [2] and VMware), to get better fault and performance isolation, improved system manageability and reduced infrastructure cost through resource consolidation and live migration [5]. 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. However, as we show later on, based on the utilization levels and characteristics of these different co-located VMs, the overall power consumption and performance of the VMs can vary a lot. This can create hotspots of activity, and degrade overall performance and energy efficiency.

In this paper, we introduce vGreen, a multi-tiered 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 (eg. 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 scheduling 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 and accordingly schedule the VMs across the PM cluster, improving the overall performance and energy efficiency by 20% and 15% respectively compared to state of the art VM scheduling policies. Furthermore, vGreen is extremely lightweight with negligible runtime overhead.

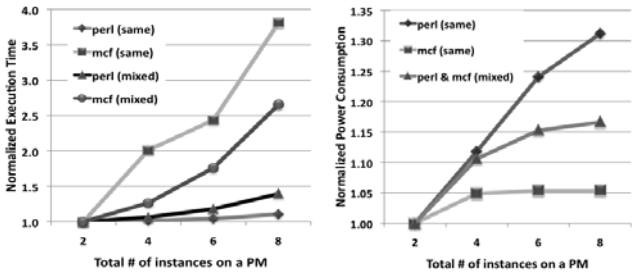
The rest of the paper is organized as follows. In section 2, we discuss the related work followed by the overall design and architecture of vGreen in section 3. We then provide the implementation details, methodology and experimental results in section 4, before concluding in section 5.

2. RELATED WORK

Systems for management of VMs across a cluster of PMs have been proposed in the past. Eucalyptus [13] and Usher [10] are open source systems, which include support for managing VM creation and allocation across a PM cluster. However, they 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 [18], the authors propose a VM scheduling system, which dynamically schedules the VMs across the PMs based on their CPU, memory and network utilization. The primary objective of the system is to avoid hotspots of activity on PMs for better overall performance. The Distributed resource scheduler

*This work has been funded in part by Sun Microsystems, UC MICRO, Center for Networked Systems (CNS) at UCSD, MARCO/DARPA Gigascale Systems Research Center and NSF Greenlight.

ISLPED'09, August 19–21, 2009, San Francisco, California, USA.



(a) Normalized Execution Time (b) Normalized Power Consumption

Figure 1: Comparison of *perl* and *mcf*

(DRS) from VMware [1], which is a proprietary solution, also uses VM scheduling to perform automated load balancing in response to CPU and memory pressure. Similarly, in [4], the authors propose VM scheduling algorithms for dynamic consolidation and redistribution of VMs for managing performance and SLA (server level agreements) violations. The authors in [7] 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 [18, 4], which can result in globally sub-optimal placement of VMs. However, none of these VM scheduling algorithms take into account the impact of the policy decisions on the energy consumption in the system.

The problem of power management in virtualized environments has also been investigated. In [12], 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 can be a potential problem, since it is difficult to port some of the state of the art power management policies like [6, 8] in guest OS because of lack of exclusive access to privileged resources such as CPU performance counters. In [14], a co-ordinated multi-level solution for power management in data centers is proposed. The model uses power estimation (using power sensors) and workload utilization levels to drive VM placement and power management. However, the model and results are based on offline trace driven analysis and simulations. Furthermore, both [12] and [14] do not take the architectural characteristics of the VM into account, which, as we show in section 3, directly determine the VM performance and power profile. In [17], 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 [6, 8], performance [9] and thermal management [11] on standalone systems has been explored before. However, in virtualized environments, it is a more complex activity due to multiple abstractions involved (virtual and physical cpus, VM, PM etc) and 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 scheduling across a PM cluster for overall performance and energy efficiency. In our knowledge, this is the first work that exploits the architectural characteristics of VMs to perform dynamic VM scheduling for general purpose workloads. 3) We implement the proposed system on a real life testbed and through extensive experiments and measurements highlight the benefits of the approach over existing state of the art VM scheduling systems.

3. VGREEN DESIGN

3.1 Motivation

The nature of workload executed in each VM determines the power profile and performance of the VM, and hence its energy consumption. In virtualized environments, VMs with different or same characteristics could be co-located on the same PM. In this section we show, that co-location of VMs with heterogeneous characteristics on PMs is beneficial for overall performance, energy efficiency and also power consumption balancing across the PM cluster.

To understand the co-relation between VM characteristics and these metrics we performed some offline experiments and analysis using benchmarks from SPEC-CPU 2000 suite, namely *mcf* and *perl*. These two benchmarks have contrasting characteristics in terms of their CPU and memory utilization. While *mcf* has high memory accesses per cycle (MPC) and low instructions per cycle (IPC), *perl* has low MPC and high IPC. We used a testbed of two dual Intel quad core Xeon based machines (eight physical CPUs each) running Xen. On each of these PMs, we created two VMs with four virtual CPUs (VCPUs) each (total of four VMs). Inside each VM we executed either *perl* or *mcf* as the workload. Since SPEC benchmarks are single threaded programs, to get results for higher processor utilization rates, we ran multiple instances of the benchmark. For our PM (eight physical CPUs), this implies two instances for 25% utilization, four instances for 50% and eight instances for 100% utilization.

In our first set of experiments, we ran homogeneous VMs on each PM, i.e. the two VMs with *mcf* on one PM, and two with *perl* on the other. During the execution, we measured the system power consumption using an AC power analyzer, and also recorded the execution time. Figure 1a 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). The results for this case are marked as 'same' indicating homogeneity. We can observe that for *mcf* (shown as 'mcf (same)'), as the number of instances increase, the execution time almost increases linearly. For eight instances of *mcf*, the execution time almost quadruples. In contrast, for *perl* ('perl (same)'), the execution time is fairly independent of the number of instances. The reason for this difference is due to the contrasting MPC of the two benchmarks. The high MPC of *mcf* results in higher cache conflict rate when multiple instances execute, which decreases its IPC and hence increases its execution time. On the other hand for *perl*, the MPC of the VM is much lower. Hence, this analysis indicates that the performance of a VM has a strong negative co-relation to MPC of the workload running inside it.

Similarly Figure 1b shows the system level power consumption of the benchmarks normalized against the power consumption with two instances. We can observe that for *perl*, as the utilization increases, the power consumption increases almost linearly, while for *mcf*, the power consumption increases initially but then it saturates. For eight instances, the difference in power consumption is almost 25%. The reason for this is the difference in their IPC. The high IPC of *perl* results in higher utilization of CPU core resources, which translates into higher CPU and system level power consumption. In contrast, *mcf* has a low IPC, which results in much lower power

consumption. This analysis indicates that the power consumption of a VM has direct co-relation to IPC of the workload running inside it.

These results indicate that co-scheduling VMs with similar characteristics is not beneficial from energy efficiency and power consumption balance point of view at high utilization rates. The PM running *mcf* contributes to higher system energy consumption (since it runs longer), while the PM running *perl* contributes to power imbalance (since it consumes higher power). To understand the benefits of co-scheduling heterogeneous workloads, we swapped two VMs on the PMs, hence running VMs with *mcf* and *perl* on both the PMs. 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 *perl* execution time almost stays the same, while *mcf* execution time goes down significantly for higher number of instances (around 150% reduction for eight instances). The average power consumption of the two PMs become identical, and is almost an average of the two PMs in the homogeneous case. Due to the high performance improvement of *mcf*, this results in system level energy savings of up to 20%.

In summary, this exercise indicates that co-scheduling VMs with heterogeneous characteristics on the same PM is beneficial from both energy efficiency and performance point of view. This is achievable in virtualized environments, since VMs can be dynamically migrated across PMs at low overhead using '*live migration*' [5]. This provides strong motivation for online characterization of VMs in the system, which has been largely unexplored in previous work as described in section 2. We next describe the overall architecture of vGreen, and present details on how it constructs VM characteristics dynamically at run time using a novel hierarchical approach.

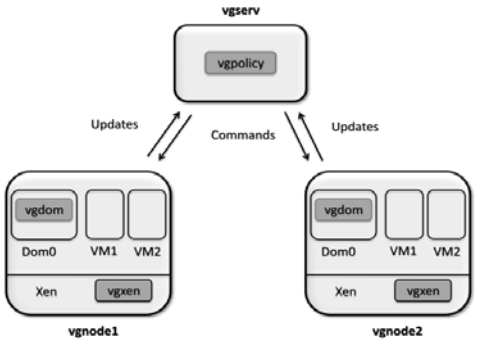


Figure 2: Overall vGreen design

3.2 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 Dom0 (*vgdom*) on each *vgnode*. We now present the details of these vGreen modules.

vgnode: A *vgnode* refers to an individual PM in the cluster. Each *vgnode* has vGreen modules (*vgxen* and *vgdom*) installed on them. The *vgxen* is a module compiled into Xen (see Figure 2) and is responsible for characterizing the running VMs. In Xen, a VM

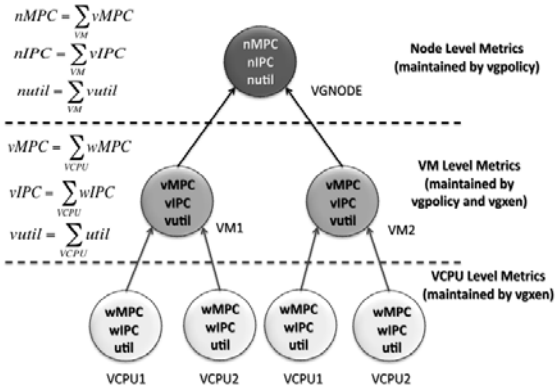


Figure 3: An example of Hierarchical Metrics in vGreen

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. A PM can have multiple VMs active on it at any point in time, and Xen multiplexes them across the real physical CPUs (PCPUs) in the machine. The entity that Xen schedules over the PCPU is the VCPU making it the fundamental unit of execution.

The estimation of MPC, IPC and utilization metrics for the VMs in vGreen is a hierarchical process. Figure 3 illustrates the hierarchy, which starts from the VCPU level, which is the actual unit of execution 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, i.e. 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 below:

$$wMPC_{cur} = MPC \cdot util$$
$$wIPC_{cur} = IPC \cdot util \quad (1)$$

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, and refer to them as weighted metrics. The equation below shows how weighted MPC is estimated:

$$wMPC = \alpha \cdot wMPC_{cur} + (1 - \alpha) \cdot wMPC_{prev} \quad (2)$$

where, the new value of weighted MPC ($wMPC$) is calculated as an exponential average of $wMPC_{prev}$, the previous value of $wMPC$, and $wMPC_{cur}$ (equation 1). 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. 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 in the system

Table 1: MPC Balance Algorithm

Input: *vgnode* $n1$

1: **if** $nMPC_{n1} < nMPC_{th}$ **then**
2: **return**
3: **end if**
4: $vm\_min \leftarrow min\_mpc\_vm(n1)$
5: $pm\_min \leftarrow NULL$
6: **for** all vgnodes n_i except $n1$ **do**
7: **if** $(nMPC_{n_i} < nMPC_{th})$ and $(nMPC_{n1} - nMPC_{th}) > (nMPC_{th} - nMPC_{n_i})$ **then**
8: **if** !pm_min or $nMPC_{pm\_min} > nMPC_{n_i}$ **then**
9: $pm\_min \leftarrow n_i$
10: **end if**
11: **end if**
12: **end for**
13: **if** pm_min and $(nMPC_{n1} - nMPC_{pm\_min}) > vMPC_{vm\_min}$ **then**
14: $do\_migrate(vm\_min, n1, pm\_min)$
15: **end if**

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 Dom0 through a shared page.

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*, and update the *vgserv* with latest data on the characteristics of different VMs running on the vgnode. Apart from this, *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.

vgserv: The *vgserv* acts as the cluster controller and is responsible for managing VM scheduling across the *vgnode* cluster. The *vgpolicy* is the core of *vgserv*, which makes the scheduling 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. 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 algorithm periodically (T_{p_period}). The basic algorithm is motivated by the fact that VMs with heterogeneous characteristics should be co-scheduled on the same *vgnode* (section 3.1). The algorithm runs in 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 1 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 1). This threshold is representative of whether high MPC is affecting the performance of the VMs in that *vgnode* (based on the observation in section 3.1). If nMPC is smaller, the function returns, since there is no MPC balancing required for $n1$ (step 2 in Table 1). If it is higher, step 4 finds the VM with the minimum vMPC in $n1$ (referred to as *vm_min*), which can be migrated to a *vgnode* with a lower nMPC than $n1$ to get a better MPC balance in the system. The target *vgnode*, to which *vm_min* can be migrated, is stored in *pm_min*, which in step 5 is initialized to NULL. In steps 6-12, the algorithm tries to find the target *vgnode* with the minimum nMPC subject to the condition in step 7. 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 step 8 and 9, 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 6 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 performs a final migration check in step 13 to confirm if the migration of *vm_min* from $n1$ to *pm_min* creates more balance in the system, and does not reverse the imbalance by making nMPC of *pm_min* more than that of $n1$. If this condition holds, then in step 14, the algorithm invokes the *do_migrate* function to live migrate *vm_min* from $n1$ to *pm_min* [5]. The decisions taken by the *vgpolicy* (updates, migration) are communicated to the *vgnodes* in form of commands as shown in Figure 2. 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 utilization of *vgnodes* to ensure there are no overcommitted nodes in the system, if there are other underutilized *vgnodes*. 4) **VM consolidation**: If all the metrics in the prior three steps are balanced, then this step tries to consolidate low utilization VMs to generate idle *vgnodes* in the system, which could be then be moved into low power states.

Table 2: Benchmarks Used

Suite	Benchmark	Characteristics
PARSEC	*freqmine*	High IPC/Low MPC
	streamcluster	Low IPC/High MPC
SPEC2K	*perl*	High IPC/Low MPC
	bzip2	High IPC/Low MPC
	equake	Low IPC/High MPC
	mcf	Low IPC/High MPC

4. EVALUATION

4.1 Implementation & Methodology

Our testbed for vGreen include two state of the art 45nm Dual Intel Quad Core Xeon E5440 (8 PCPUs each) based server machines, which act as the *vgnodes*, and a Core2Duo based desktop machine that acts as the *vgserv*. The *vgnodes* run Xen3.3.1, and use Xeno-Linux 2.6.18 for Dom0. The *vgxen* module is implemented as part of the Xen credit scheduler (the default scheduler) to record VM metrics and exposes a new hypercall to map the shared page for sharing VM metrics data with *vgdom* (as explained in section 3). The *vgdom* module is implemented in two parts on Dom0: 1) A Linux driver that interfaces with *vgxen* to get the VM characteristics and exposes it to the application layer, 2) An application client module that gets the VM metrics data from the driver and passes it on to *vgserv*. It also accepts *vgserv* commands and processes it. vGreen requires no modifications to the VMs running on a node.

The *vgserv* runs on Linux 2.6.23, and is implemented as an application server module. On initialization, it 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/consolidation decisions as described in section 3.

For our experiments, we use benchmarks with varying characteristics from the SPECCPU 2000 and PARSEC [3] suites. PARSEC benchmarks are parallel and multi-threaded, while SPEC benchmarks are single threaded. The used benchmarks and their characteristics are illustrated in Table 2. We can observe that *freqmine*, *perl* and *bzip2* are high IPC/low MPC CPU intensive benchmarks, while *streamcluster*, *mcf* and *equake* are low IPC/high MPC memory intensive benchmarks. We run each of these benchmarks inside a VM, which is initialized with four VCPUs and 0.5GB 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 AC power analyzer.

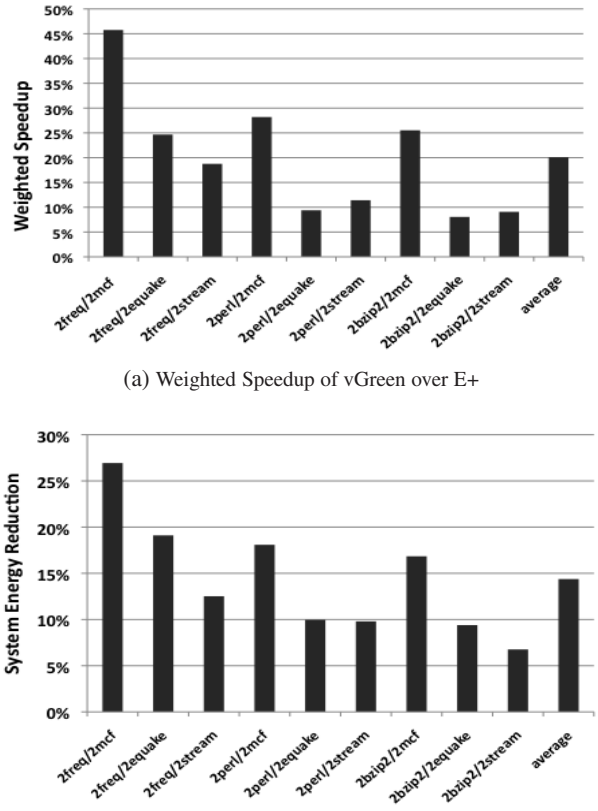
We compare vGreen to a VM scheduler that mimics the Eucalyptus VM scheduler [13] for our evaluation. Eucalyptus is an open source cloud computing system, that can manage VM creation and allocation across a cluster of PMs. The default Eucalyptus VM scheduler assigns VMs using a greedy policy, i.e. it 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 utilization metrics and utilization/consolidation algorithms proposed in the previous section, which allow it to redistribute/consolidate VMs dynamically at run-time. This enhancement is representative of the metrics employed by the existing state of the art policies (see section 2). We refer to this enhanced scheduler as E+. Furthermore, we use the same initial assignment of VMs to the PMs as done by the E+ scheduler for vGreen as well.

We report the comparative results of vGreen and E+ for three parameters: 1) **Energy savings:** We estimate the energy reduction in executing each combination of VMs using vGreen over E+. This is calculated by measuring the total 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) **Weighted Speedup:** We also estimate the average speedup of each VM combination with vGreen. For this, we use the weighted speedup (WS) based on a similar metric defined in [16]. It is defined as:

$$WS = \frac{\sum_{VM_i} \frac{T_{e+_i}}{T_{alone_i}}}{\sum_{VM_i} \frac{T_{vgreen_i}}{T_{alone_i}}} - 1 \quad (3)$$

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. To calculate WS, we normalize T_{e+_i} and T_{vgreen_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 equation 3. WS > 0 implies that the VM combination runs faster with vGreen and vice versa. 3) **Reduction in Power Imbalance:** We also estimate the % reduction in power imbalance in the system with vGreen due to IPC balancing compared to E+. We estimate this using %reduction in variance in power consumption of the two *vgnodes*. A high value for this metric indicates better power balance across the two *vgnodes* with vGreen.

For all our experiments, we use P_{p_period} and P_{up_period} as 5s. In our experience, this was frequent enough to detect and resolve imbalances in the system for different workload combinations with minimal runtime overhead. 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 comfortably separate high MPC and high IPC VMs from each other.



(a) Weighted Speedup of vGreen over E+

(b) Energy Savings of vGreen over E+

Figure 4: Energy Savings and Weighted Speedup comparison

4.2 Results

Heterogeneous Workloads: In the first set of experiments, we use combinations of VMs running benchmarks with heterogeneous characteristics. Each VM consists of four instances (for SPEC) or four threads (for PARSEC) of the benchmark. In total we run four VMs, which make both the *vgnodes* close to 100% utilized. We run high IPC benchmarks in first two VMs, and high MPC benchmarks in the other two.

Figure 4 shows the results for weighted speedup and energy savings. The x-axis on the graphs shows the initial distribution of VMs on the physical machines. For instance, *2freq/2mcf* means that two VMs running *freqmine* are on the first PM, while the two VMs running *mcf* are on the second. We can observe in Figure 4a, that vGreen achieves an average of around 20% weighted speedup over E+ across all the combinations. 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 close to 100%, 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 allow the *vgserv* to detect a heavy MPC imbalance. This initiates migration of 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 balance avoids unnecessary cache conflicts, and results

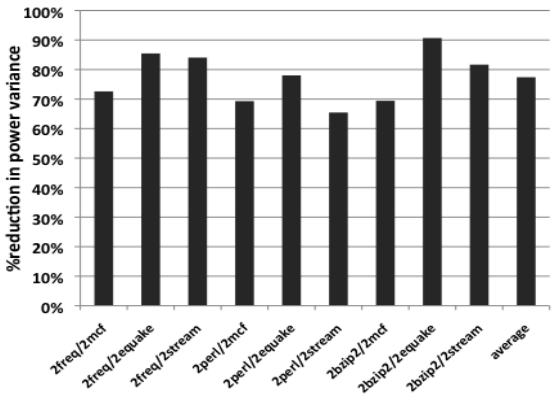


Figure 5: % Reduction in Power Imbalance

in significant speedup of the high MPC VMs as observed in Figure 4a. We can see in Figure 4a, that some combinations achieve higher weighted speedup compared to others. For instance, for *2freq/2mcf* combination is it over 45%, while for *2perl/2mcf* it is around 30%. This difference is due to the fact that co-location of *perl* and *mcf* VMs results in a small slowdown of *perl*, while co-location of *freqmine* and *mcf* has no impact on the execution time of *freqmine*. This results in higher weighted speedup for *2freq/2mcf* compared to *2perl/2mcf* (refer to equation 3). The speedup also results in lower system energy consumption, since now the benchmarks run and consume active power for a smaller duration. We can see in Figure 4b that vGreen results in average system energy savings of around 15% across all the benchmark combinations. We can observe some combinations to have more savings than the others, which is primarily due to their different weighted speedups.

Figure 5 illustrates the balance in power consumption across the two *vgnodes* achieved using vGreen over E+. The figure shows the %reduction in power consumption variance across the *vgnodes* to capture the degree of balance achieved. We can see that vGreen reduces the average power variance across the two *vgnodes* by close to 80%. This happens due to the better balance of IPC and utilization across the machines with vGreen, which results in well balanced and similar power consumption for the *vgnodes* compared to E+. This results in a better overall thermal and power profile and reduces power hotspots in the cluster.

Homogeneous Workloads: We also experimented with combination of VMs running homogeneous benchmarks. We did experiments for all the six benchmarks in Table 2, where all the fours 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. Consequently, the results for all the three parameters were the same as that for E+ across all the experiments.

In summary, based on these experiments, we conclude that vGreen performs significantly better than E+, when the VM characteristics vary, and does as well as E+, when they are identical.

4.3 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 no overhead on software execution and 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 3 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 with and without *vgxen* and *vgdom*. We also observed negligible overhead of VM migration on execution times of benchmarks, which is consistent with the findings in [5]. Across all our experiments, we had an average of around two VM migrations for each VM combination run, which is a very small overhead (< 1%) compared to the overall timeframe.

5. CONCLUSION

In this paper we presented vGreen, a system for energy efficient computing in virtualized environments. The key idea behind vGreen is linking workload characterization to VM scheduling decisions to achieve better performance, energy efficiency and power balance in the system. We designed novel hierarchical metrics to capture VM characteristics, and develop simple balancing policies to achieve the above mentioned benefits. We implemented vGreen on a real life testbed of state of the art server machines, and show with SPEC and PARSEC benchmarks, that it can achieve improvement in performance and system level energy savings by up to 20% and 15% respectively over state of the art policies.

6. REFERENCES

[1] VMware Dynamic Resource Scheduler, http://www.vmware.com/files/pdf/drs_datasheet.pdf.

[2] P. Barham, B. Dragovic, K. Fraser, S. Hand, T. Harris, A. Ho, R. Neugebauer, I. Pratt, and A. Warfield. Xen and the art of virtualization. In *Proc. SOSP '03*.

[3] C. Bienia, S. Kumar, J. P. Singh, and K. Li. The PARSEC benchmark suite: characterization and architectural implications. In *Proc. PACT '08*.

[4] N. Bobroff, A. Kochut, and K. Beaty. Dynamic placement of virtual machines for managing SLA violations. In *Proc. International Symposium on Integrated Network Management '07*.

[5] C. Clark, K. Fraser, S. Hand, J. G. Hansen, E. Jul, C. Limpach, I. Pratt, and A. Warfield. Live migration of virtual machines. In *Proc. NSDI'05*.

[6] G. Dhiman and T. Rosing. System-level power management using online learning. *IEEE Transactions on CAD'09*.

[7] F. Hermenier, X. Lorca, J.-M. Menaud, G. Muller, and J. Lawall. Entropy: a consolidation manager for clusters. In *Proc. VEE'09*.

[8] C. Isci, G. Contreras, and M. Martonosi. Live, runtime phase monitoring and prediction on real systems with application to dynamic power management. In *Proc. MICRO'06*.

[9] R. Knauerhase, P. Brett, B. Hohlt, T. Li, and S. Hahn. Using OS observations to improve performance in multicore systems. *IEEE Micro'08*.

[10] M. McNett, D. Gupta, A. Vahdat, and G. M. Voelker. Usher: an extensible framework for managing custers of virtual machines. In *Proc. LISA'07*.

[11] A. Merkel and F. Bellosa. Balancing power consumption in multiprocessor systems. *SIGOPS Oper. Syst. Rev. '06*.

[12] R. Nathuji and K. Schwan. VirtualPower: coordinated power management in virtualized enterprise systems. In *Proc. SOSP '07*.

[13] D. Nurmi, R. Wolski, C. Grzegorczyk, G. Obertelli, S. Soman, L. Youseff, and D. Zagorodnov. The Eucalyptus open-source cloud-computing system. In *Proceedings of Cloud Computing and Its Applications'08*.

[14] R. Raghavendra, P. Ranganathan, V. Talwar, Z. Wang, and X. Zhu. No "power" struggles: coordinated multi-level power management for the data center. In *Proc ASPLOS'08*.

[15] P. Ranganathan, P. Leech, D. Irwin, and J. Chase. Ensemble-level power management for dense blade servers. In *Proc. ISCA '06*.

[16] A. Snavely and D. M. Tullsen. Symbiotic jobscheduling for a simultaneous mutlithreading processor. *In Proc. ASPLOS'00*.

[17] A. Verma, P. Ahuja, and A. Neogi. Power-aware dynamic placement of HPC applications. In *ICS '08*.

[18] T. Wood, P. J. Shenoy, A. Venkataramani, and M. S. Yousif. Black-box and gray-box strategies for virtual machine migration. In *Proc. NSDI'07*.