Energy-Efficient Server Consolidation for Multi-threaded Applications in the Cloud

Can Hankendi
ECE Department
Boston University, Boston, MA
Email: hankendi@bu.edu

Ayse K. Coskun
ECE Department
Boston University, Boston, MA
Email: acoskun@bu.edu

***Abstract*—Cloud services have been actively used for transactional and batch workloads. Recently, multi-threaded high-performance computing (HPC) workloads have started to emerge on the cloud as well. Unlike most traditional data center loads, HPC workloads highly utilize the servers. The energy efficiency and performance of HPC loads, however, vary strongly as a function of the amount of allocated resources. This paper proposes an autonomous resource allocation technique for multi-threaded compute-intensive HPC workloads with the goal of creating tunable energy cost-performance tradeoffs for the cloud administrators and users. The proposed technique adjusts the available resources for the virtual machines (VMs) based on application energy efficiency while delivering the desired performance guarantees. Experiments on a real-life multi-core server show that the proposed technique improves the system throughput-per-watt by 17% on average compared to existing techniques.**

I. Introduction

Energy-related costs are among the major contributors to the total cost of ownership in today's data centers and HPC clusters [1]. As a result, energy-efficient operation is one of the prerequisites in achieving sustainable computing. Another important challenge for large computing clusters arises from the fact that as the number of servers increase, cluster management incurs higher complexity. In fact, server management costs have increased 3 times from 2000 to 2012 [1]. This increase motivates the design of automated energy and resource management policies.

While many modern data centers running enterprise workloads successfully implement energy and resource management techniques today, multi-threaded workloads are emerging as new candidates for the cloud. As the hardware resources are designed to provide high levels of parallelism, HPC domain, as well as an increasing number of applications in other domains, employ multi-threaded applications to efficiently utilize the underlying hardware resources. Some of these multi-threaded HPC-type loads are expected to leverage the cloud resources, as the cloud provides comparable performance to native (not virtualized) systems for HPC workloads [5]. As a result, cloud providers (e.g., Amazon, SGI) have already started providing HPC resources for their customers. In addition to delivering high performance, cloud offers cost-efficient and highly elastic computing resources compared to the traditional grid infrastructures [6].

Virtualized cloud resources also provide opportunities to improve the energy efficiency through server consolidation.

978-1-4799-0623-9/13/$31.00 © 2013 IEEE

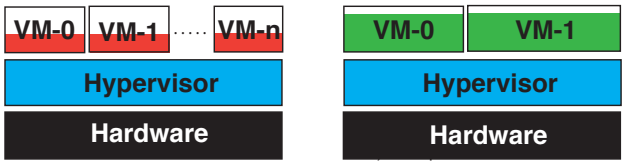

Fig. 1. Virtualized server illustration for a) enterprise workloads and b) HPC-type workloads.

Consolidating workloads on the same physical node reduces the number of active servers and increases the utilization of individual nodes, improving the energy efficiency. However, the energy efficiency of a consolidated system considerably varies depending on the potential resource contentions on the physical node. An additional challenge with multi-threaded loads is the significance of application-specific needs and constraints while determining the resource allocation strategies [7]. Multi-threaded workloads also differ from enterprise loads, as the system utilization is generally higher even in absence of consolidation.

In Figure 1, we show two potential virtualization scenarios for a typical multi-core server node. In Figure 1.a, we illustrate a server node that predominantly runs transactional workloads with low CPU utilization. For such cases, consolidating server nodes with a high VM density (number of VMs per physical CPU) is a common practice. On the other hand, in Figure 1.b, we illustrate the virtualization case for HPC-type multi-threaded workloads, which highly utilize the CPU resources. Thus, the number of VMs per node in HPC clouds is expected to be much fewer than the traditional cloud setting. The higher utilization trends together with the multi-threaded application characteristics (e.g., scalability) make the decisions about *how much resource to allocate per application* a crucial part of consolidation. In addition, providing energy-efficient consolidation strategies for multi-threaded loads enables tuning the energy cost and performance of the applications. For example, through consolidation, administrators can offer a wider range of cost-performance tradeoffs to the users, who then could select the most desirable operating point for their applications.

In this paper, we propose a novel, autonomous resource allocation strategy for consolidating multi-threaded HPC-type workloads on multi-core servers. While some of the prior approaches targeting multi-threaded loads select which applications to co-schedule together for reducing resource contention (e.g., [3], [8]), we find that performance isolation in VMs reduces the significance of co-runner application

selection. Thus, our technique focuses on adaptively deciding how much resource to allocate for each VM in a consolidated environment. The performance change due to increasing or decreasing the amount of resources depend on the application characteristics. Therefore, our technique favors the applications that benefits more from additional CPU resources to improve the energy efficiency of the server node. Our technique uses runtime performance polling to identify application phases. Our specific contributions are as follows:

- Performance isolation in consolidated environments minimizes the impact of a co-runner application on the other applications' performance and achieves better performance predictability. We analyze the performance isolation on consolidated virtual environments. We show that the virtual environments provide comparable performance and on average 60% higher performance isolation in comparison to native OS environments.
- Following our observations in performance isolation, we propose a runtime policy that makes resource allocation decisions proportionally to the energy efficiency of the applications. For the PARSEC benchmark suite [9] running on a multi-core server, we show that our technique can be jointly used with application-selection policies to improve the throughput-per-watt by 17% on average and up to 21% compared to using only co-runner application selection policies.
- To provide performance guarantees to the user under consolidation, we propose a feedback technique that adjusts the resource allocation decisions to meet the performance constraints. We demonstrate that our policy is able to seamlessly track application phases and varying resource needs while maintaining the desired performance.
- We provide a practical implementation strategy for our technique. We implement and evaluate the energy efficiency and performance of the proposed approach on a real multi-core based virtualized server.

The rest of the paper starts with an overview of the related work in virtualized system management. Section III provides the details of our experimental setup on a real-life server. Section IV evaluates application selection based co-scheduling policies and investigates performance isolation in consolidated environments. Section V explains the proposed resource allocation policy and the runtime implementation details. Section VI presents the experimental results and Section VII concludes the paper.

II. Related Work

A number of the existing energy and resource management techniques target optimization of an entire computing cluster. We call such techniques "cluster-level". For **power management of clusters**, Fan *et al.* study power provisioning strategies for large scale data centers [10]. Wang *et al.* propose a feedback controller to optimize the cluster-level performance while meeting the power constraints [11]. Nathuji *et al.* propose a VM-aware power allocation technique to improve performance for a given power budget [12]. Their proposed technique allocates power budgets proportionally across VMs by favoring applications that are service level agreement (SLA) critical.

Cluster-level resource management techniques can be divided into two groups: VM migration and consolidation techniques. The goal of cluster-level resource management techniques is mainly to provide service availability guarantees (e.g., [13], [14]). Consolidation and migration policies target balancing the activity on various server components such as CPU, memory, or disk to improve energy efficiency (e.g, [15]). Modern virtualization environments such as Xen, KVM and vSphere provide resource management mechanisms to improve the efficiency of the server nodes mainly through scheduling techniques. While KVM relies on default Linux resource management, Xen and vSphere provide management options such as Xen Management Tools and vSphere's Distributed Resource Scheduler (DRS), which mostly rely on VM migration to provide balanced load distribution across server nodes [21], [22]. Zheng *et al.* present an empirical infrastructure for data center management [16]. Their proposed infrastructure allocates a server node (i.e., sandbox) to experimentally derive the energy/performance tradeoffs. Wang *et al.* propose a framework that allows user-specified workload provisioning policies to optimize energy efficiency on clusters [17]. Their framework allocates threads to available cores across the cluster depending on the user-specified performance/power constraints. Vasic *et al.* propose the *DejaVu* framework that makes resource allocation decisions based on the history of the VMs to reduce the resource management overhead [18].

Node-level techniques (i.e., on a single server) provide more visibility into workload characteristics, and this visibility can be used to improve the overall energy efficiency of the clusters. The goal of node-level techniques is reducing resource contention by pairing contrasting applications or threads to work on the same physical resources (e.g., [19], [3]). Bhadauria *et al.* propose co-scheduling algorithms based on tracking application characteristics such as cache misses and bus contentions [4]. Their method determines the time and space (e.g., number of cores) share of the co-scheduled applications. The authors propose algorithms to find the best time and space sharing based on offline energy-delay measurements. Dhiman *et al.* propose a VM scheduling technique that estimates VM-level CPU and memory usage based on system-level metrics to guide scheduling and migration decisions [8]. The success of their technique depends on identifying the workloads that have complementary characteristics. Their technique, however, does not consider adjusting the resource allocations for the co-scheduled applications.

Our resource sharing technique differentiates from the previous work in the following aspects. We propose a novel resource allocation strategy for consolidated multi-threaded HPC workloads, whereas most of the prior work focuses on traditional data center loads. We demonstrate that the proportional allocation of the shared resources improves the energy efficiency by taking the varying application characteristics into account. Our technique is able to identify the applications that benefit more from increasing CPU resources and favors them when making resource allocation decisions. In contrast to previous work, our technique dynamically makes resource allocation decisions without requiring offline analysis. We implement our technique on a real-life server and show that our technique can be jointly used with application-selection policies to improve the energy efficiency of multi-threaded workloads.

III. Experimental Methodology

In this section, we present the details of our experimental methodology. Our target environment is a multi-core system that is similar to commonly used servers in virtualized HPC clusters and data centers. As our main focus in this work is HPC-type workloads, so our experimental setup is similar to the case in Figure 1.b.

We perform all experiments on an AMD 12-core Magny Cours (Opteron 6172) server, virtualized by the VMware vSphere 5.1 ESXi hypervisor. Magny Cours is a single-chip processor that comprises two 6-core dies attached side by side. Each core has a 512 KB private L2-cache. Each 6-core die has one local NUMA node and a 6 MB shared L3-cache. All cores share a 16 GB off-chip memory. Although we perform our experiments on a single-chip processor, our approach can be generalized to multi-chip (i.e., multi-socket) servers as well as to multiple server nodes, as discussed in Section V-D. For each co-scheduled application, we create a separate VM on top of the hypervisor. Therefore, in all of our experiments, number of VMs is equal to the number of co-scheduled applications and all VMs run Ubuntu Server 12.04 as the guest OS. As multi-threaded workloads are expected to run on multiple cores and scale well to at least 4 threads/cores, most of the results and analyses are performed on a system with 2 VMs.

Performance counters, which are available in today's processors, provide visibility into the system and application characteristics. It is possible to poll the performance counter data from the guest OS by utilizing the virtualized performance counters in vSphere 5.1. We use `perf` utility tool on the guest OSes to poll the following performance counter data from the physical CPUs at every second: CPU cycles, retired instructions, and L3-cache misses. We find these as the most relevant metrics to determine the performance and power characteristics of the applications, in line with findings of prior work [7], [20]. We use `esxtop` utility to collect VM-level resource utilization data such as CPU, memory and disk utilization at every 2 seconds from the vSphere hypervisor. We configure `esxtop` to report data only for the VMs to reduce the runtime overhead, as opposed to reporting the data for all processes running on the hypervisor. We measure system power by using a *Wattsup PRO* power meter with a 1 second sampling rate, which is the minimum sampling rate provided for this meter. As the total system power determines the electricity cost of a server, we evaluate system power rather than component power (i.e., processor, disk, etc.).

We run PARSEC [9] multi-threaded benchmarks in our experiments as a representative set of multi-threaded workloads. We run each benchmark with 12 threads using the native input set. `fluidanimate` and `facesim` require 2^n number of threads to run. Thus, we run these two benchmarks with 16 threads.

Parallel applications typically consist of serial I/O stages and a parallel phase, i.e., region-of-interest (ROI). As ROI is the power and performance hungry portion of parallel applications, it is important to consider only the ROI phase for evaluating the energy and performance tradeoffs during consolidation. In fact, parallel phases of the multi-threaded workloads occupy most of the compute cycles of the processors in real-life HPC systems. As the start and end points of ROI phase vary across different applications, we implement a consolidation management interface, `consolmgmt`, that synchronizes the ROIs of the co-scheduled applications. We implement the ROI-Synchronization routine inside the existing PARSEC `HOOKS` library. The VM that first reaches the ROI phase sleeps until the second VM reaches its own ROI. The second VM sends interrupts upon reaching ROI to resume the execution and start data logging. We stop data collection and terminate the applications after one of the applications reaches the end of its ROI phase. PARSEC `HOOKS` library is only used as the markers for the parallel phases. For other applications, it is also straightforward to detect the parallel phase by monitoring the overall utilization of the system, without requiring additional libraries.

Benchmark Set	CPU	Memory
3x canneal, 3x ferret, 2x bodytrack, dedup, vips	Low	High
4x blackscholes, 2x vips, bodytrack, freqmine, streamcluster, swaptions	High	Low
3x bodytrack, 3x facesim, 2x fluidanimate, canneal, streamcluster	Medium	Medium

TABLE I. Benchmark sets and their relative CPU and memory intensities. Each benchmark set consists of 10 PARSEC benchmarks.

In order to evaluate existing techniques, we design three benchmark sets that exhibit distinct CPU and memory characteristics. Each benchmark set consists of 10 PARSEC benchmarks, which would occupy 5 server nodes when consolidated. In Table 1, we list the benchmark names, number of instances of each benchmark and CPU and memory characteristics of each benchmark set. To evaluate our technique in a cluster setting, we also generate 50 random workload sets in a similar way to the benchmark sets in Table 1.

IV. Co-scheduling Analysis in Virtualized Environments

This section investigates the impact of co-scheduling on application performance under virtual and native OS (i.e., Linux) environments. The goal is to develop an understanding of the tradeoffs and constraints in virtualized environments to enable better policy design for runtime management. We first evaluate performance and performance variation on both virtual and native OS environments. We then investigate whether selecting which applications to co-schedule together on the same resources changes the overall energy efficiency on virtual systems.

A. Performance Isolation on Virtualized Environments

Performance isolation is a desirable feature in consolidated environments, as poor performance isolation leads to unpredictable performance for the applications sharing the same resources. In order to quantify the performance and the performance isolation on virtual environments, we first compare performance (i.e., throughput) and performance variation on native OS and virtual environments. We utilize throughput (retired instructions per second) as the performance metric, as it is a good indicator of the application progress and can be measured without instrumenting the applications. For ensuring that the throughput is a meaningful metric to track the performance of PARSEC applications, we perform correlation analysis, which shows that runtime and throughput are strongly correlated (i.e., 0.98 Pearson coefficient with 0.99 confidence

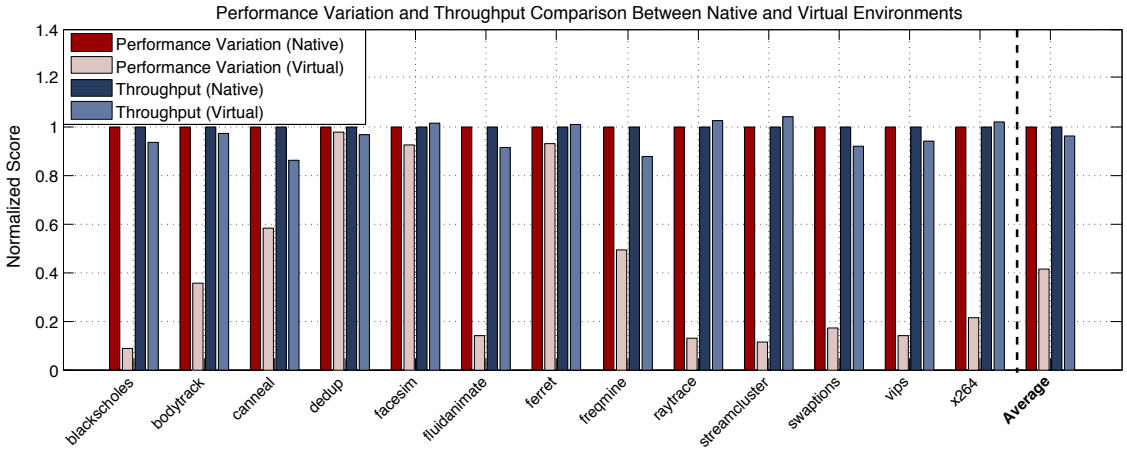

Fig. 2. In this experiment, a PARSEC benchmark is co-scheduled with another benchmark (only two benchmarks at a time). The experiment is repeated to cover all possible application pairings. Figure shows the performance variation (standard deviation/mean) and the average throughput of each benchmark across its co-scheduled runs with the other benchmarks. We report the performance and performance variation normalized with respect to the native environment. Smaller performance variation indicates better performance isolation.

level). It is also possible to measure application-specific performance metrics (e.g., frames/second (FPS)) by instrumenting the applications. We demonstrate the runtime behavior of our runtime policy with *application-specific constraints* in Section VI.

For all the experiments on the native OS, we co-schedule 2 applications at a time, each of them running with 6 threads. For the virtual system, we create 12 vCPUs (12 threads) per VM and distribute the total CPU resources equally across VMs. In Figure 2, we report the normalized performance variation and throughput values for all the PARSEC benchmarks with respect to native environment. As Figure 2 shows, on average virtual environment provides 60% lower performance variation, which implies that the application performance is significantly less affected by the co-runner application, indicating higher performance isolation. Virtualization causes less than 4% overhead compared to the native OS environment. As a result, the energy efficiency improvements due to application selection policies are expected to be limited on virtual environments with high performance isolation capabilities.

B. Application Selection Based Co-scheduling

Co-scheduling the application pairs that have contrasting performance characteristics, such as co-scheduling a high instructions-per-cycle (IPC) application with a low IPC application, is expected to improve the energy efficiency significantly, as it leads to more balanced resource usage. Based on this fact, application selection based co-scheduling techniques are proposed to improve the energy efficiency [8], [4]. Application selection based co-scheduling techniques first rank the applications according to a selected metric and then co-schedule the highest ranked benchmark with the lowest one, and proceed through the ranked list in a similar fashion. Energy efficiency improvements due to application selection policies rely on the fact that co-runner applications affect each other's performance. However, virtual environments provide more isolated execution environment for individual VMs (i.e., applications), which limits the benefits of application selection based co-scheduling policies.

In order to quantify the energy efficiency improvements due to application selection policies, we evaluate the *throughput-per-watt* for the workload sets shown in Table 1. We use throughput-per-watt as a metric of energy efficiency, as it captures the useful work done per watt consumed [4]. Recall that throughput is a meaningful estimator for application progress for PARSEC workloads.

We implement previously proposed co-scheduling policies that determine the best application pairs to co-schedule together by ranking applications using two metrics (i.e., *Memory per cycle (MPC)*CPU Utilization* and *IPC*CPU Utilization*) [8]. We also evaluate the throughput-per-watt of the workload sets when applications to be co-scheduled are selected randomly. Figure 3 compares the energy efficiency of the three distinct benchmark sets under various co-scheduling policies. *IPC*CPU Utilization* provides the best results for the medium and high CPU benchmark sets, whereas *MPC*CPU Utilization* is the best policy for the highly memory intensive benchmark set. However, on average, best performing policy improves the energy efficiency by 4% in comparison to the random policy.

Although application selection policies improve the energy efficiency of the virtual systems, these improvements are limited for a system with high performance isolation motivating the design of novel resource management techniques to further improve the energy efficiency of server nodes running multi-threaded applications. We next describe our autonomous

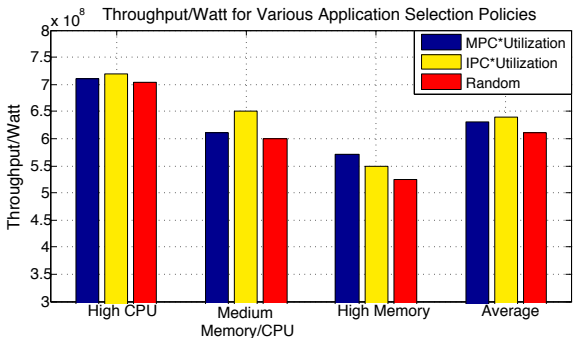

Fig. 3. Throughput-per-watt for the benchmark sets in Table 1, when co-scheduled using previously proposed application-selection policies. On average, randomly co-scheduling applications provides comparable energy efficiency in comparison to previously proposed policies.

resource allocation policy, which aims to improve the energy efficiency through energy proportional resource allocation for multi-threaded workloads.

V. Adaptive Resource Sharing for Multi-threaded Workloads

This section presents our adaptive resource sharing technique for multi-threaded workloads. Our technique maximizes the energy efficiency of a server node by allocating resources to VMs based on application-specific energy and performance characteristics. The goal is to provide more resources to energy-proportional applications, whose performance improves as an increasing amount of CPU resources are allocated to the application. Our resource sharing technique dynamically adjusts the amount of CPU resources allocated for each VM by evaluating the relative energy proportionality of the co-scheduled applications.

A. Resource Management in Virtual Environments

For server level resource management, modern hypervisors provide resource control knobs to the administrators to manage the resources allocated for VMs. During the VM creation process, initially CPU, memory and disk resources are allocated for each VM. Due to varying resource requirements, these initially configured resources can be reconfigured through the hypervisor at runtime, without any need for restarting the VMs. ESXi hypervisor provides various features such as vCPU hot plugging, and adjusting resource reservations, limits and shares. Reservation can be defined as the guaranteed minimum resource allocation that is always available to the VMs and resource limits restrict the resource usage of the VMs. By adjusting the reservations and/or the limits, it is possible to control the resource usage of individual VMs to optimize the performance and power tradeoffs. In this work, we propose running each VM with 12 vCPUs and using the CPU resource limits settings for resource allocation to be able to utilize all of the 12 physical CPUs (pCPU) when desired.

Running a larger number of vCPUs might introduce higher overhead on the hypervisor side, as the hypervisor needs to handle a higher number of vCPUs and multiplex them to run

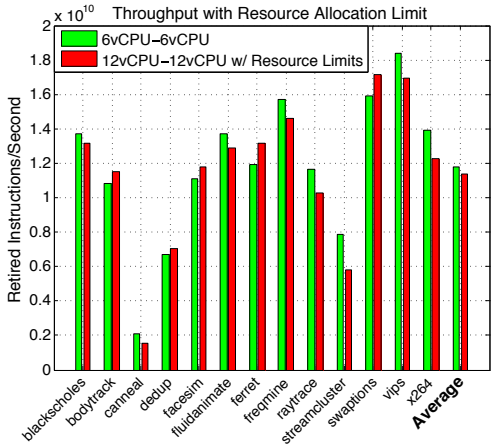

Fig. 4. Average throughput of each PARSEC benchmark when they are co-scheduled with all other benchmarks separately in a two-VM configuration (i.e., 12 vs. 6 vCPUs). Creating VMs with 12 vCPUs brings less than 2% overhead, in comparison to VMs with 6 vCPUs.

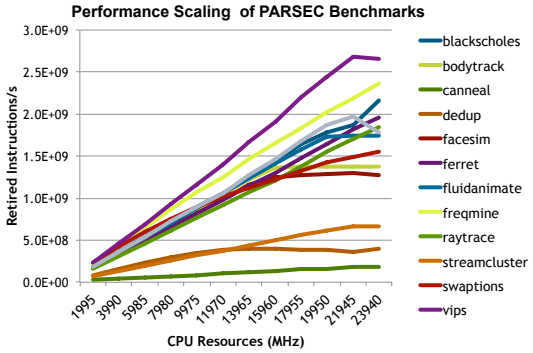

Fig. 5. Throughput scaling of PARSEC benchmarks as a function of CPU resource limits.

on the pCPUs. In order to quantify the overhead of running a higher number vCPUs on the performance of the applications, we compare two scenarios. In the first scenario we create 2 VMs with 6 vCPUs. In the second scenario we create 2 VMs with 12 vCPUs each and we limit the resource allocation of each VM to the level equal to running 6 vCPUs. In Figure 4, we show the average throughput (retired instructions per second) for all PARSEC benchmarks, when each benchmark is co-scheduled with all the others separately (i.e., 2 application at a time). As Figure 4 shows, running a higher number of vCPUs with resource limitations introduces less than 2% overhead on average in comparison to VMs with 6 vCPUs.

B. Autonomous Resource Sharing

To improve the energy efficiency of the system, we propose to proportionally allocate the resources depending on the energy efficiency levels of each co-scheduled application by utilizing the VM resource management knobs (i.e., CPU resource limits). In Figure 5, we show the throughput of all PARSEC benchmarks as a function of CPU resource limits. As Figure 5 shows, the impact of changing the CPU resource limits on application performance varies significantly depending on the application characteristics. For instance, the throughput of `vips` increases dramatically as more CPU resources are provided, while the throughput increase of `canneal` is minimal. Following our observations, we propose to improve the energy efficiency by favoring applications that benefit more from increased CPU resources, such as allocating more CPU resources to `vips`, when it is co-scheduled with `canneal`.

To find the resource share of each VM at runtime, we first compute a weight, w_i, for each application, based on the ratio of their throughput values, w_i, such that $w_i = t_i / \sum_{i=1}^{n} t_i$. We then use w_i to allocate (r_i) amount of resources for VM_i, where $r_i = w_i * R$, and R is the total amount of available resources. On the ESXi environment, available resources are represented in units of frequency, f (MHz). We allocate CPU resources (r_i) for VMs, such that $\sum_{i=1}^{n} r_i = 23940MHz$, where 23940 MHz (12 pCPUs) is the maximum available CPU resources (R) for the VMs on our 12-core system.

C. Consolidation with Performance Constraints

Allocating fewer resources to the applications that are not energy-efficient have negative impacts on the throughput of some workloads. However, in a real-life scenario, users might request performance guarantees for their specific applications.

In addition, administrator might want to consider other constraints, such as *fairness*, to achieve long-term scheduling goals. To be able to provide performance guarantees to the users, we implement a feedback mechanism into the resource allocation routine. In our implementation, users can set either *maximum throughput degradation* constraints or *application-specific performance constraints*, such as minimum frames per second, to guarantee a certain level of performance for their applications.

Our resource allocation routine first stores the applications' performance in a lookup table (LUT), when both applications have equal resources, as a reference value to calculate the gains and loss due to changing CPU resource allocations. At runtime, the resource allocation routine continuously monitors the performance changes on each application by comparing the current performance of the application with respect to the stored value in the LUT. The feedback mechanism takes the maximum performance degradation as a user input, and asserts a signal to the resource allocation routine, if there are any performance violations. To implement the application-specific performance constraints, we utilize the Heartbeats API to monitor application performance in PARSEC [23]. In this case, the feedback mechanism communicates with the Heartbeats API to monitor the application-specific performance instead of monitoring the throughput. Feedback mechanism is able to send separate signals specific to each VM (i.e., *VM0-alert*, *VM1-alert*), therefore it is possible to enforce performance constraints on individual consolidated applications. When the resource allocation routine receives a feedback signal, it increments and decrements the CPU resources at the frequency (MHz) granularity that is equivalent to 1 pCPU, to meet the performance constraints. However, adjusting the CPU resources at a finer granularity is also possible.

D. Runtime Implementation

We implement our autonomous resource sharing technique on an AMD Magny Cours multi-core server (see Section III). Our architecture consists of a management node (vCenter terminal) and virtualized server(s). Figure 6 shows the architecture of our implementation. Utilizing a centralized management node is a common practice on VM environments (e.g., VMware's vCenter Server). VMware's VM management framework uses SDKs and APIs, some of which are leveraged in our implementation. In general, data center administrators do not always have access to the hypervisor code and management through a centralized node brings ease of implementation. Our technique, however, could be implemented within the hypervisor as well for open-source hypervisors.

In a multiple node scenario, management node still serves as the centralized resource manager. Each host (server node) is then interfaced to the management node through the default vSphere SDK. Thus, extension to multiple nodes and/or to multi-socket chips requires no major modification to the implementation. In this work, we demonstrate the capabilities of our runtime implementation on a single server node.

We use an Intel i3 dual-core processor based machine as our management node, which runs Ubuntu 12.04 as its OS. Management node periodically collects runtime performance statistics from the ESXi hypervisor. The runtime monitor polls

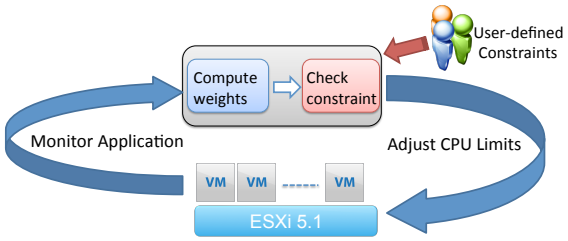

Fig. 6. Runtime operation of the resource allocation technique.

VM-level performance counter readings (i.e., retired instructions, clock cycles) every second. The management node then makes resource allocation decisions and the resource allocation routine communicates with the ESXi through the vSphere SDK to perform administrative tasks (i.e., VM reconfiguration) on VMs.

VI. Experimental Results

In this section, we first present the runtime behavior of the resource allocation technique on a real-life server. We then show the capabilities of the feedback mechanism that enables user to enforce performance guarantees for selected applications. We then report average throughput-per-watt improvements compared to the existing co-scheduling policies.

A. Runtime Behavior

Figure 7 demonstrates the runtime behavior of our technique for the application pair, `blackscholes` and `raytrace` under *throughput degradation constraint*. Similarly, in Figure 8, we show the application pair `bodytrack` and `x264`, under application-specific performance constraints (i.e., minimum frames per second (FPS)). In both figures, we show the progress (at the top) of each VM after both applications reach their parallel phases, and the CPU resource limits (at the bottom), which are imposed by our method. Initially, each application is allocated equal resources and then resource allocation decisions are enforced every second proportionally to the throughput of the applications. User-defined constraints are represented as dashed lines on the plot. We test our technique with a *maximum throughput degradation* constraint of 30% on `raytrace` for the application pair in

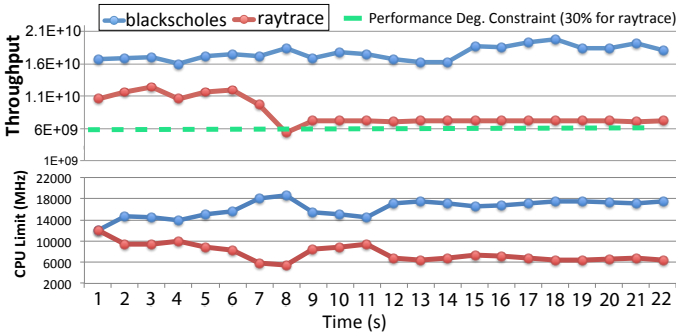

Fig. 7. Runtime behavior of the resource allocation routine with throughput constraints. CPU resources are adjusted proportional to throughput of the applications to improve the overall efficiency of the server. At t=9, resource allocation routine responds to the performance violation that occurred at t=8, by increasing the CPU resources allocated to `raytrace` to meet the performance constraints.

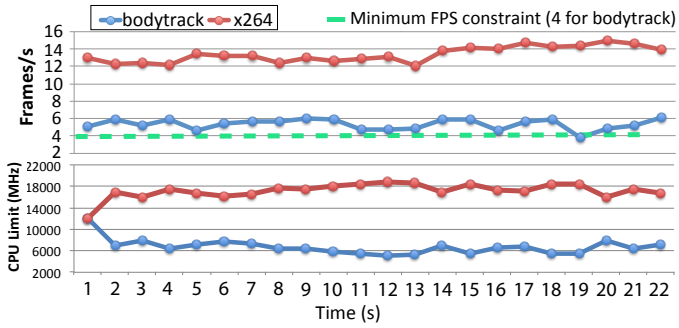

Fig. 8. Runtime behavior of the resource allocation routine with application-specific performance guarantees. In this case, we test our technique with a minimum FPS constraint of 4 on `bodytrack`. Our technique adapts the resource allocation at t=19 following the performance constraint violation.

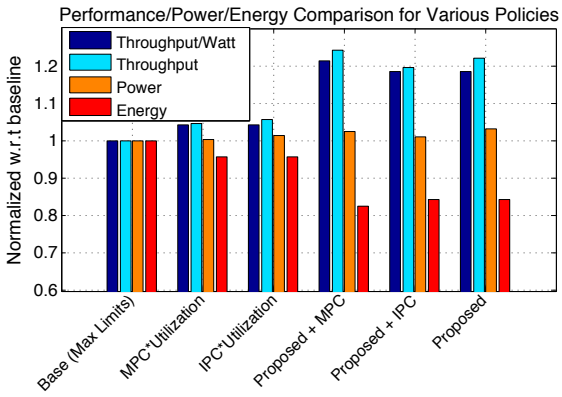

Fig. 9. Average throughput-per-watt, throughput, power and energy comparison for randomly generated 50 workload sets, normalized w.r.t baseline case of assigning each VM unlimited CPU resources.

Figure 7, and *minimum FPS* constraint of 4 on `bodytrack` in Figure 8. We measure the number of frames processed per second (FPS) as the performance metric for two of the video processing applications (i.e., `bodytrack`, `x264`) in the PARSEC suite.

In Figure 7, at t=8, performance of `raytrace` falls below the maximum throughput degradation limit. Therefore, feedback mechanism signals the resource allocation routine to increase the CPU resources allocated to `raytrace`, and to reduce the CPU resources allocated to `blackscholes`. Similarly, in Figure 8, at t=19, FPS for `bodytrack` falls to 3.78, therefore the resource allocation are adjusted to meet the minimum FPS constraint by increasing the CPU resources allocated to `bodytrack`. The ability to fine tune the performance as shown in these examples enables cloud providers to offer flexible service prices with varying performance guarantees. At the same time, by maintaining target performance at runtime, the proposed method enables users to run their time-sensitive jobs as part of a consolidated system.

In the results presented so far, we synchronize the parallel phases of the applications through the `consolmgmt` interface that is explained in Section III. However, our implementation works seamlessly as applications go in and out of parallel phases and does not disrupt the default scheduler decisions. If we consider the entire execution of the applications including the serial I/O phases, throughput-per-watt gains are much higher in comparison to the ROI execution, as we have higher gains when one application is in serial and the other is in its parallel phase. For instance, throughput-per-watt improvements for `blackscholes-raytrace` pair reaches 24% for the entire execution and 11% for the ROI execution compared to the baseline case without any resource controls. For all of the other results, we report energy efficiency improvements for the ROI only, as energy gains during parallel phases are more valuable in real-life settings.

B. Evaluation for Various Cluster Workload Sets

To evaluate impact of our technique on the overall energy efficiency of a cluster in a real-life scenario, we generate 50 random workload sets, each containing 10 PARSEC benchmarks as in the three workload sets described in Section IV. For each workload set, we evaluate the application selection based co-scheduling policies (i.e., using *MPC*Utilization* and *IPC*Utilization* metrics), the proposed technique and the combination of application selection policies and the proposed approach (i.e., *Proposed+IPC*, *Proposed+MPC*).

Figure 9 compares the average throughput-per-watt, throughput, power and energy consumption of the workload sets for various techniques. We normalize the values with respect to the baseline case, where VMs do not have any limits on CPU resources. The proposed technique alone improves the throughput by 21% with 3% increase in power consumption, which translates into 16% lower energy consumption with respect to the baseline case. Moreover, our proposed policy can further improve the energy efficiency by 17%, when jointly utilized with application selection policies, while the application selection policies alone improves the energy efficiency by only 4%. The other important observation from our results is the fact that energy efficiency improvements are resulting from achieving increased throughput at a marginal power increase. Therefore, our runtime policy lowers the total energy consumption for executing all 50 workload sets by 14% in comparison to only using application-selection based co-scheduling policies.

C. Consolidation with a Higher Number of VMs

We test our research allocation technique for co-scheduling 3, 4 and 6 applications (i.e., each application on a separate VM) on the same server. As a case study, we first create a set of 12 applications (2x blackscholes, 2x dedup, 2x vips, bodytrack, canneal, facesim, swaptions, streamcluster, x264). In each experiment, we co-schedule the applications in groups of 2, 3, 4 or 6 applications at a time. We evaluate our resource sharing policy with various numbers of VMs by allocating the CPU resources proportionally to the throughput of the applications. We compare the energy efficiency improvements with respect to the baseline case (i.e., without any limits on CPU resources). Figure 10 shows that the 2-VM case achieves 16% energy efficiency improvements on average with respect to the baseline case, whereas the 3-VM case improves the energy efficiency by 9%. The energy efficiency improvements decrease with increasing the number of applications co-scheduled at a time.

As the HPC-type multi-threaded applications already utilize the resources at high levels, increasing the VM density diminishes the energy efficiency improvements, leaving less

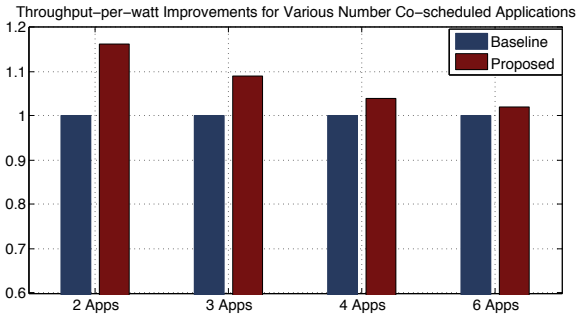

Fig. 10. Normalized throughput-per-watt w.r.t. the baseline case, where each VM is given the maximum resources, for a varying number of co-scheduled applications. Energy efficiency improvements decrease as we increase the number of co-scheduled VMs.

room for managing the performance/energy tradeoffs. In addition, resource contention at lower levels of cache is expected to be higher when a larger number of vCPUs share the hardware resources. In our experiments, we test our technique under a fixed amount of CPU resources (i.e., single-node). Therefore, it is expected to observe lower gains with an increasing number of VMs, as the performance of all the PARSEC applications scale well up to 4 threads. Even though our method works with an arbitrary number of VMs and server nodes, each multi-threaded application should be allocated a sufficient amount of CPU resources to ensure high performance.

VII. Conclusions

Energy efficiency remains to be a major challenge for computing clusters. As multi-threaded workloads start to leverage cloud resources, efficient consolidation of these workloads emerge as a novel research area. In this paper, we have evaluated existing co-scheduling techniques that are based on co-runner application selection. Our work shows that in the case of multi-threaded loads running on multi-core systems, it is more important to adjust the allocated resources depending on the power efficiency of the applications compared to solely selecting which applications to co-schedule. This result is mainly due to the performance isolation advantages of the virtualized environments.

Following our analysis, we have presented a novel policy for autonomous resource allocation for multi-threaded loads. Our policy proportionally allocates the resources according to energy efficiency of the applications to efficiently utilize the server node. Our technique includes a feedback mechanism to set user-defined performance targets per application. Based on our experiments on a real-life server, our policy achieves 17% higher throughput-per-watt on average compared to the state-of-the-art co-scheduling techniques.

Acknowledgment

This work has been partially funded by VMware, Inc. and MGHPCC seed funds.

References

[1] M. Bailey, "The Economics of Virtualization: Moving Toward an Application-Based Cost Model," *International Data Corporation (IDC), Whitepaper*, December 2009.

[2] L. A. Barroso and U. Hölzle, "The Case for Energy-Proportional Computing," *IEEE Computer*, pp. 33–37, 2007.

[3] X. Meng, C. Isci, J. Kephart, L. Zhang, E. Bouillet, and D. Pendarakis, "Efficient Resource Provisioning in Compute Clouds via VM Multiplexing," in *International Conference on Autonomic Computing (ICAC)*, 2010, pp. 11–20.

[4] M. Bhadauria and S. A. McKee, "An Approach to Resource-aware Co-scheduling for CMPs," in *ACM International Conference on Supercomputing*, 2010, pp. 189–199.

[5] C. Macdonell and P. Lu, "Pragmatics of Virtual Machines for High-performance Computing: A Quantitative Study of Basic Overheads," in *International ACM Symposium on High-Performance Parallel and Distributed Computing (HPDC)*, 2008, pp. 141–152.

[6] Vecchiola, C. and Pandey, S. and Buyya, R., "High-performance Cloud Computing: A View of Scientific Applications," in *International Symposium on Pervasive Systems, Algorithms, and Networks (ISPAN)*. IEEE, 2009, pp. 4–16.

[7] R. Cochran, C. Hankendi, A. K. Coskun, and S. Reda, "Pack & Cap: Adaptive DVFS and Thread Packing Under Power Caps," in *International Symposium on Microarchitecture (MICRO)*, 2011, pp. 175–185.

[8] G. Dhiman, G. Marchetti, and T. Rosing, "vGreen: A System for Energy-efficient Computing in Virtualized Environments," in *Proceedings of International Symposium on Low PowerElectronics and Design (ISLPED)*, 2009, pp. 243–248.

[9] C. Bienia, "Benchmarking Modern Multiprocessors," Ph.D. dissertation, Princeton University, January 2011.

[10] X. Fan, W. dietrich Weber, and L. A. Barroso, "Power Provisioning for a Warehouse-sized Computer," in *In Proceedings of International Symposium on Computer Architecture (ISCA)*, 2007, pp. 13–23.

[11] X. Wang and M. Chen, "Cluster-level Feedback Power Control for Performance Optimization," in *International Symposium on High Performance Computer Architecture (HPCA)*, 2008, pp. 101–110.

[12] R. Nathuji, K. Schwan, A. Somani, and Y. Joshi, "VPM Tokens: Virtual Machine-aware Power Budgeting in Datacenters," *Cluster Computing*, pp. 189–203, 2009.

[13] N. Bobroff, A. Kochut, and K. Beaty, "Dynamic Placement of Virtual Machines for Managing SLA Violations," in *International Symposium on Integrated Network Management.*, 2007, pp. 119 –128.

[14] A. Beloglazov and R. Buyya, "Adaptive Threshold-based Approach for Energy-efficient Consolidation of Virtual Machines in Cloud Data Centers," in *International Workshop on Middleware for Grids, Clouds and e-Science*, 2010, pp. 1–6.

[15] A. Merkel, J. Stoess, and F. Bellosa, "Resource-conscious Scheduling for Energy Efficiency on Multicore Processors," in *European Conference on Computer Systems (EuroSys)*, 2010, pp. 153–166.

[16] W. Zheng, R. Bianchini, G. J. Janakiraman, J. R. Santos, and Y. Turner, "JustRunIt: Experiment-based Management of Virtualized Data Centers," in *USENIX Annual Technical Conference*, 2009, pp. 18–18.

[17] W. Wang, T. Dey, R. W. Moore, M. Aktasoglu, B. R. Childers, J. W. Davidson, M. J. Irwin, M. Kandemir, and M. L. Soffa, "Reeact: A customizable virtual execution manager for multicore platforms," in *Virtual Execution Environments*, 2012, pp. 27–38.

[18] N. Vasić, D. Novaković, S. Miučin, D. Kostić, and R. Bianchini, "Dejavu: Accelerating resource allocation in virtualized environments," in *Architectural Support for Programming Languages and Operating Systems (ASPLOS)*, 2012, pp. 423–436.

[19] R. L. Mcgregor and C. D. Antonopoulos, "Scheduling Algorithms for Effective Thread Pairing on Hybrid Multiprocessors," in *International Parallel and Distributed Processing Symposium (IPDPS)*, 2005, p. 28.

[20] M. Khan, C. Hankendi, A. Coskun, and M. Herbordt, "Software Optimization for Performance, Energy, and Thermal Distribution: Initial Case Studies," in *Green Computing Conference and Workshops (IGCC)*, 2011, pp. 1 –6.

[21] "Xen Management Tools," http://wiki.xen.org/wiki/Xen_Management_Tools.

[22] "Resource Management with VMware DRS," http://www.vmware.com/pdf/vmware_drs_wp.pdf.

[23] H. Hoffmann, S. Sidiroglou, M. Carbin, S. Misailovic, A. Agarwal, and M. Rinard, "Dynamic Knobs for Responsive Power-aware Computing," in *ASLPOS*, 2011, pp. 199–212.