vCap: Adaptive Power Capping for Virtualized Servers

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

Sherief Reda
School of Engineering
Brown University, Providence, RI
sherief_reda@brown.edu

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

***Abstract*—Power capping on server nodes has become an essential feature in data centers for controlling energy costs and peak power consumption. More than half of the server nodes are virtualized in today's data centers; thus, providing a practical power capping technique for consolidated virtual environments is a significant research problem. This paper proposes a power capping technique, vCap, which makes resource allocation decisions to maximize the Quality-of-Service (QoS) while meeting the power constraints in virtualized servers that run multi-threaded applications. For a given set of applications, vCap first decides which applications to co-schedule based on application scalability and then optimizes the QoS in an application-aware manner for each VM by adaptively adjusting the CPU resources. Experiments on real-life multi-core servers show that vCap provides 12% higher energy efficiency in comparison to the state-of-the-art power capping techniques, while adhering to the power cap 92% of the time within a 2W error margin.**

I. Introduction

The total cost of ownership of modern data centers is increasingly dominated by the power and cooling costs. As the computational demands continue to grow in the cloud era, efficient management of power consumption is even more critical to enable sustainable operation of the data centers. Constraining the peak power consumption of the servers in data centers has become a common practice for managing the energy costs and to comply with the power delivery limitations [1]. In addition, wholesale energy markets offer new incentives for dynamic regulation of data center power. Specifically, it is possible to significantly reduce the energy costs by closely following the power consumption signals provided by the independent system operators, who need to match power supply and demand in the grid [2]. Providing fine granularity power capping capabilities for servers is a necessary step in designing data centers that can benefit from such opportunities in the emerging energy markets.

Virtualizing the data center resources has become another common practice in modern data centers. Virtualization reduces the hardware and power requirements by enabling seamless consolidation of multiple applications on a smaller set of physical resources. In addition, virtualization provides features such as performance isolation, flexible resource management, and virtual machine (VM) migration across physical servers to facilitate energy-efficient operation. As a result, the number of virtualized servers started to outnumber the native (not virtualized) servers in recent years [3]. Designing dynamic power capping techniques that meets the QoS demands in virtualized servers is, therefore, a significant research problem.

This paper proposes a dynamic power capping technique for virtualized servers, specifically targeting multi-threaded applications. As virtualized cloud environments provide comparable performance to native execution, multi-threaded workloads from various applications domains (e.g., high performance computing, scale-out applications) are commonly deployed in virtualized data centers. VMs that run multi-threaded workloads, Symmetric Multi-Processing (SMP) VMs, exhibit significantly different power-performance tradeoffs compared to the VMs that run traditional enterprise loads with low system utilization. In addition, the energy efficiency of SMP VMs varies because of the multi-threaded application characteristics such as inter-thread communication and performance scaling. Therefore, optimizing the performance of the virtualized servers under power constraints requires sufficient understanding of the application characteristics [4].

The proposed technique, **vCap**, monitors the performance characteristics of the VMs and dynamically adjusts the resource allocation decisions to improve the energy efficiency of the virtualized server nodes, while adhering to the power caps. Our specific contributions are as follows:

- We propose a co-scheduling technique that finds the best VM pairs to consolidate together based on the scalability of the multi-threaded applications to maximize the achievable performance. We show that scalability-based co-scheduling outperforms prior co-scheduling methods that solely consider application resource use.
- We propose a fine-grained power capping technique, **vCap**, that is able to meet the performance objectives such as maximizing the total QoS or achieving a minimum QoS level for specific VMs. We also propose a fast and accurate runtime QoS estimation methodology for the VMs without requiring any offline training. Based on the QoS estimations, **vCap** distributes the resources among VMs to optimize the energy efficiency of the server node.
- We demonstrate the benefits of **vCap** on a real-life multi-core server. **vCap** is able to meet dynamically changing power constraints with high accuracy while improving the overall QoS provided to the user by 17% and the QoS/watt by 12% in comparison to the state-of-the-art techniques for workload sets created out of PARSEC [5] and Cloudsuite [6] benchmarks.

The rest of the paper starts with an overview of the related work in power capping. Section III explains our experimental setup. Section IV discusses multi-threaded application characteristics that are relevant to the design of our policy. Section V

explains the details of **vCap**. Section VI presents our results on a real-life server and Section VII concludes the paper.

II. Related Work

Most of the modern processor cores support dynamic voltage-frequency scaling (DVFS) and power gating capabilities. Therefore, DVFS and core power gating have become traditional power management knobs [7]. Recent commercial servers also provide power capping capabilities [8]. For example, Intel Sandybridge provides a power estimator and a runtime average power limiter (RAPL) [9].

Raghavendra *et al.* propose a global power management technique for clusters to coordinate the power provisioning for individual nodes under power constraints [10]. For multi-threaded applications, Rangan *et al.* propose thread scheduling policy that maps the threads to various voltage-frequency (V-F) domains to optimize performance under power constraints. Reda *et al.* propose a runtime controller to meet the peak power constraints through DVFS and packing threads onto a smaller number of cores, while optimizing the application performance [11]. Ma *et al.* propose a power capping technique by power-gating the cores and applying per-core DVFS for a mixture of single and multi-threaded applications running on native servers [12].

For capping the power in virtualized environments, Nathuji *et al.* design a power allocation technique for VMs to improve the performance for a given power budget [1] by allocating power budgets proportionally across VMs according to the service level agreement (SLA) requirements of individual VMs. The proposed technique uses CPU utilization data to distribute the available power budget. Dhiman *et al.* propose a VM scheduling technique that estimates VM-level CPU and memory usage based on system-level metrics to guide co-scheduling and migration decisions [4]. Their proposed technique consolidates the applications that have complementary resource usage characteristics to reduce the performance degradation. Hwang *et al.* study the impact of CPU consolidation in virtualized multi-core environments [13]. Their study investigates finding the optimum VM-density for multi-core processors for single-threaded applications that have distinct characteristics (i.e., memory/CPU-bounded) and they propose a consolidation policy that uses DVFS and core power gating. Vasic *et al.* introduce the *DejaVu* framework that makes resource allocation decisions based on the history of the VMs to reduce the resource management overhead [14]. Another line of work in VM resource management targets reducing the resource contention to improve the efficiency of the virtualized servers [15], [16].

Our power capping technique, **vCap**, differentiates from previous work in the following aspects. Our technique takes the performance scalability of the applications into account while making co-scheduling and resource allocation decisions to maximize the server energy efficiency. While most of the prior work in consolidation focuses on application selection during co-scheduling, our technique dynamically makes resource allocation decisions to maximize the performance under power and performance constraints. Our analysis shows that co-scheduling multi-threaded workloads solely based on their resource use is not sufficient to maximize QoS under power constraints. **vCap** achieves finer granularity power tracking compared to existing DVFS or clock gating based methods, making it a promising technique for future data centers with dynamic power regulation capabilities.

III. Experimental Methodology

In this work, we target multi-core based servers that run multi-threaded applications. Our experimental setup includes a server containing an AMD 12-core Magny Cours (Opteron 6172) processor, virtualized by the VMware vSphere 5.1 ESXi hypervisor. Magny Cours consists of two 6-core dies attached together on a single chip. Each 6-core die includes a 12 MB shared L3 cache, and each core has a 512 KB private L2 cache. All cores also share a 16 GB off-chip memory. We create VMs with multiple vCPUs (SMP VMs) such that each VM accommodates a multi-threaded application. Each VM is initialized with 12 vCPUs as 12 is the maximum possible number of vCPUs for a system with 12-cores. Each VM runs Ubuntu 12.04 as the guest OS.

We run 7 applications from the PARSEC multi-threaded benchmark suite [5] and 3 applications from Cloudsuite [6] in our experiments as a representative set of multi-threaded workloads on the cloud resources. We track the application-specific performance metrics for the PARSEC benchmarks by utilizing the Application Heartbeats framework [17]. CloudSuite applications report application-specific performance without requiring a modification to the source code. We choose to evaluate application-specific performance metrics, as instruction count based performance metrics often do not provide a meaningful performance feedback to the user, since the performance (i.e., QoS) metric differs depending on the application. For instance, for image processing applications (e.g., `bodytrack`) the QoS metric is *frames-per-second* (FPS), whereas the QoS metric for the option trading application (e.g., `blackscholes`) is the *number of options*. Instead of such metrics, application-specific performance (e.g., frames per second, requests serviced per second, etc.) can be tracked at the cost of minimal modifications to the application source code. We report the relative QoS for each application, where we define the maximum QoS of an application (i.e., QoS=1) as the case where the application is running alone with the maximum amount of available CPU resources (e.g., maximum number of cores).

To measure VM-level CPU metrics, we utilize the `esxtop` utility that is available in the ESXi hypervisor. We sample the VM-level metrics every 2 seconds, which is the minimum sampling rate for `esxtop`. We measure the system power by using a *Wattsup PRO* power meter with a 1 second sampling rate. As the total system power determines the electricity cost of a server, we focus on the system power rather than the component power (i.e., processor, disk, etc.). The resource allocation decisions are handled by the hypervisor and the OS-level tools, as described in Section V.

In all of our experiments, we only evaluate the parallel phases of the applications, as the parallel phase of multi-threaded applications dominates the application execution time in real-life clusters. We implement a consolidation management module, that synchronizes the starting point of the parallel phases of the applications and collect performance data only for the synchronized parallel phases.

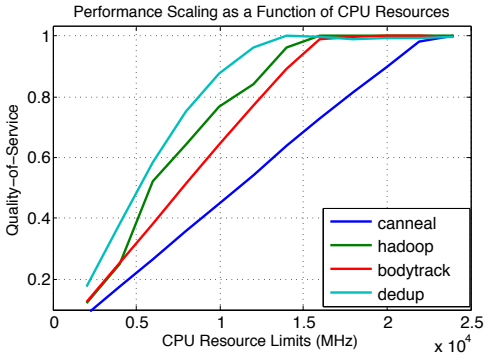

Fig. 1. Performance scaling of some of the PARSEC and benchmarks and `hadoop` as a function of CPU resource limits.

IV. Motivation

Ideally, multi-threaded applications are designed to efficiently utilize an arbitrary number of cores. However, application characteristics such as inter-thread communication and architectural bottlenecks such as the off-chip bus bandwidth cause sub-linear performance improvements when the amount of CPU resources is increased. SMP VMs that run *poorly scaling* applications are not able to utilize all the available CPU resources of a multi-core system; hence, such VMs are good candidates for consolidation. Although consolidation might degrade the performance of the individual VMs, the aggregate performance of the server node and the energy efficiency can improve significantly.

On the ESXi hypervisor, the total available computational capacity of a server node is represented in MHz, where the total amount of CPU resources, R, is equal to the number of physical CPUs multiplied by the maximum core frequency. CPU resource usage of a VM can be constrained by adjusting the *CPU resource limits* on the ESXi hypervisor. Figure 1 shows the QoS scaling of 4 applications from PARSEC and Cloudsuite as a function of the CPU resources (in MHz). For example, as Figure 1 shows, `bodytrack` cannot utilize all the available hardware resources, therefore the QoS does not improve beyond a certain amount of CPU resources (i.e., 15970 MHz). In addition, reducing the CPU resources has a larger performance impact on the poorly scaling VMs (e.g., canneal, bodytrack) at lower CPU resource limits.

Total QoS of a consolidated server depends on the specific VMs that are co-scheduled. Figure 2 shows the QoS breakdown of two systems, each of which are running two distinct VMs under various power caps. We observe that the overall QoS improvement of consolidation is much larger for

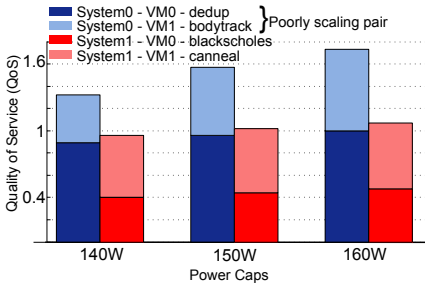

Fig. 2. Overall normalized QoS of two distinct co-scheduling cases under various power caps. QoS range for the *scaling VMs* is much smaller than the *non-scaling VMs*. Thus, selecting non-scaling VMs to co-schedule have high potential for energy efficiency improvement.

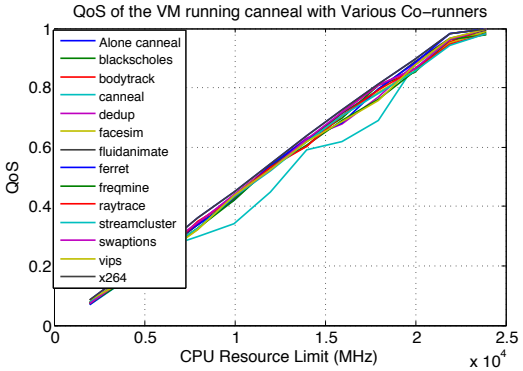

Fig. 3. QoS of the VM running `canneal` when consolidating with all the other VMs in pairs of two. Performance of `canneal` is not significantly affected by any of the co-runners.

the VMs running poorly scaling applications (i.e., blue bars) in comparison to scaling applications, as these VMs are not able to fully utilize the system when running alone. For the VMs running `dedup` and `bodytrack`, consolidation improves the overall QoS by 73% and QoS/watt by 68% under a 160W power cap. Consolidating the SMP VMs that have near-linear performance improvements as the amount of resources grow (i.e., VMs running `blackscholes` and `canneal`) provides 7% higher total QoS and only 4% higher QoS/watt compared to running each application alone. This observation motivates the design of a co-scheduling policy that takes the scalability characteristics of the applications into account while making co-scheduling and resource allocation decisions.

Consolidating VMs that have complementary resource usage characteristics is expected to reduce the resource contention. For example, memory-boundedness can be used as a metric to choose which VMs to co-schedule [4]. Depending on the requirements of the applications, it is known that consolidation causes performance degradation due to increased contention on the bus and the caches [4]. However, on virtual environments it is possible to minimize the impact of the co-runner application through isolating and balancing the memory accesses. In order to demonstrate that, we evaluate the impact of the co-runner VMs on performance of the most memory-intensive application (i.e., `canneal`). Figure 3 shows the QoS of the VM running `canneal` alone and consolidating with all other VMs in pairs of two VMs at a time. Consolidating two instances of `canneal` has the highest negative impact on the performance. However, in all other cases co-runner applications do not introduce significant performance degradation. Our experimental results discussed above imply that when consolidating multi-threaded applications, performance scalability has a more dominant impact on the energy efficiency of the server. Figure 4 shows the scalability and the memory-boundedness metrics for a selection of PARSEC and Cloudsuite benchmarks. As Figure 4 shows, memory-boundedness and scalability have significantly different trends (i.e., 0.34 Pearson coefficient with 0.26 confidence level) across benchmarks. Therefore, co-scheduling decisions based on memory-boundedness would differ from decisions that are based on the application scalability. Based on our analysis, we make the following observations:

- When consolidating multiple VMs, it is beneficial to allocate only the necessary resources to poorly scaling

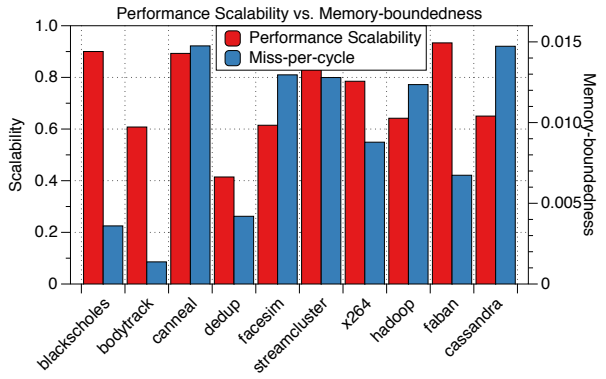

Fig. 4. Memory-boundedness (last level cache misses per cycle) vs. scalability of PARSEC and Cloudsuite applications. Scalability is measured as the ability to utilize the 12-core system when running with 12 threads. This experiment shows that memory-boundedness does not capture the scalability characteristics of the applications.

VMs that reach nearly their maximal QoS with a relatively smaller amount of resources (e.g., dedup), and to reserve the remaining resources for the co-runner VMs.

- As the performance of the VMs are minimally affected by the co-runner VMs, consolidation decisions based on resource usage (e.g., memory-boundedness) are not sufficient to improve the energy efficiency. Scalability of a VM is a more important factor while making consolidation decisions when compared to resource usage metrics.

V. Adaptive Power Capping

In this section, we discuss the design of *vCap* and provide an overview of our implementation. We first discuss our methodology to track the power caps using the CPU resource limits and our VM-level QoS estimation technique under power caps. We then present our consolidation and the resource distribution algorithm that maximizes the energy efficiency under power caps. Virtual environments provide additional control knobs to manage the amount of resources allocated for each VM. ESXi hypervisor allows the administrator to limit the maximum amount of CPU resources allocated to a specific VM by adjusting the *CPU resource limit* knob, which restricts the resource usage of the VMs. By adjusting the CPU resource limits, it is possible to cap the peak power consumption of the server node.

A. Dynamic Power Capping

By utilizing the CPU resource limits, it is possible to adjust resources with a MHz granularity, instead of adjusting the number of active threads and/or DVFS. Therefore, CPU resource limit knob enables us to control the performance/power tradeoffs with a finer granularity. In order to quantify our observation, we run experiments with power capping through resource limits vs. DVFS and number of threads. Our results show that, power capping with adjusting resource limits provides up to 14% higher QoS in comparison to power capping with adjusting DVFS and active number of threads.

In this work, we propose to execute applications with the number of threads that is equal to number of cores and then applying CPU resource limits to meet the power caps, as dynamically changing the thread number requires modification to the original code. An alternative solution is packing the threads onto smaller number of cores, which is also proposed previously to meet the power caps [11]. However, running a higher number of threads may introduce performance overheads due to increased contention and communication among threads. In order to quantify the negative impact of running a higher number of threads, we compare the normalized runtime of the applications in three cases. In the baseline case, we execute the applications with 4 threads, where there is no overhead due to higher number of threads. We then test two techniques: (1) *Resource Limits* represents the case of running the application with 12 threads and limiting CPU resources equal to the compute power of 4 cores; (2) *GuestPacking* represents the case where VMs run with 12 threads, which are packed onto 4 vCPUs at the guest OS level using thread affinity settings. Figure 5 shows the normalized runtime for running the applications with the configurations explained above. GuestPacking reduces the number of vCPUs that simultaneously access the pCPUs. As a result, GuestPacking allows us to reduce the overhead by 45% for `dedup`. Therefore, we implement GuestPacking together with CPU resource limit adjustments to minimize the negative impact of running a higher number of threads under power caps.

In order to accurately track the power caps, our proposed technique utilizes runtime power monitoring feedback. We express the total power consumption P_{tot} as the sum of the dynamic (P_{dyn}) and idle (P_{idle}) power. At runtime, **vCap** estimates the total amount of available CPU resources (R_{cap}), that meets the given power cap. For n number of VMs consolidated, $R_{cap}(MHz) = Utilization * P_{cap}/P_{dyn}$, where utilization is the percentage of the active cycles of a processor over the total number of cycles. Based on the calculated R_{cap}, we first derive the number of active vCPUs, such that $\#ofvCPUs * CoreFreq > R_{cap}$. We then reduce the CPU resource limits to be equal to the R_{cap} value.

B. Estimating QoS Degradation Under Power Caps

In order to estimate the QoS degradation due to power constraints, we utilize the VM-level metrics provided by the ESXi hypervisor (i.e., `RUN%`, `READY%`, `COSTOP%`, `WAIT%`). In this work, we focus on `READY` and `RUN` to identify the CPU demand level of the applications and the QoS of the applications at various CPU resource limits.

`RUN`: The percentage of total scheduled time of the VM, which excludes the system time (`%UTIL` = `%RUN` + `%SYS`).

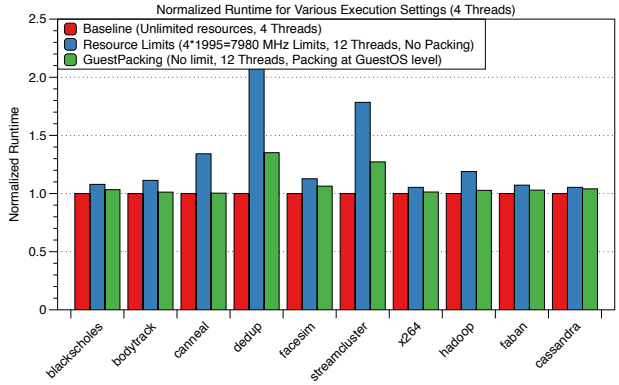

Fig. 5. Performance overhead for running higher number of threads under power caps. Running applications with 12 threads and applying resource limits introduce large overheads for some of the applications. Packing the threads onto a smaller number of vCPUs reduces the overhead by up to 45%.

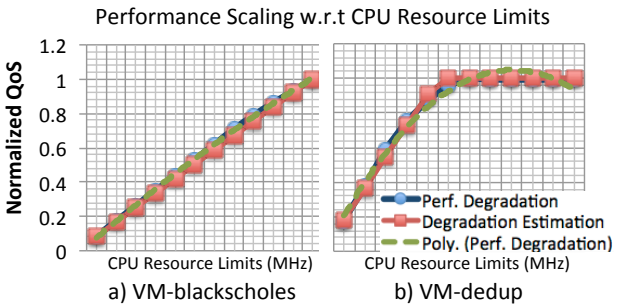

Fig. 6. QoS as a function of CPU resource limits for `blackscholes` and `dedup`. Degradation estimations are derived using the Equation 1. Equation 1 provides accurate QoS prediction without requiring any offline/training phase when compared to polynomial models.

`READY`: The percentage of time that the VM is ready to run, but not scheduled. This metric implies that the application would be able to utilize the CPU if more resources were allocated to the VM. Therefore, `READY` metric can be utilized to estimate maximum utilization level that an application can reach, which reflects the performance scalability characteristics of the applications.

It is possible to estimate the total CPU requirements of a VM by using the `RUN` and `READY` metrics. Although `RUN` metric is similar to CPU utilization metric, the additional `READY` metric captures the potential performance improvements of the VM, if more CPU resources were allocated. The total CPU requirement of an application (C_{req}) can be estimated by $C_{req}(MHz) = (\texttt{RUN\%} + \texttt{READY\%}) * R$, where R is the total amount of resources available in the processor. For a given CPU resource limit (C_{lim}), performance degradation of an application with respect to the highest CPU resource limit (i.e., QoS=1) can be calculated as,

$$QoS(C_{lim}) = \begin{cases} 1 - \frac{(C_{req} - C_{lim})}{C_{req}} & C_{req} > C_{lim} \\ 1 & C_{req} < C_{lim} \end{cases} \quad (1)$$

It is also possible to train a workload specific model for QoS estimation, however this requires additional training and offline data collection time for each new workload. Considering the vast number of diverse application sets that are running on the cloud, offline training may incur large overhead. Figure 6 shows actual and predicted QoS for `blackscholes` and `dedup`, as a function of CPU resource limits. *Degradation estimation* reflects the QoS estimations using Equation 1, while we use a second order polynomial fit as the baseline estimation technique that requires an offline training phase for each application. For all applications, Equation 1 provides 97% accuracy in QoS estimation, while polynomial fit provides 95% accuracy. These results show that the proposed QoS estimation technique provides higher accuracy even without any training phase, and eliminates the workload characterization overhead.

Providing QoS Guarantees: As decreasing the amount CPU resources allocated to a VM causes degradation on the application performance, it is essential to provide lower-bounds for the QoS of individual VMs. In a real-life scenario, users might request minimum QoS guarantees (QoS_{req}) for time-sensitive applications. In order to provide QoS guarantees for the VMs, we can use the Equation 1 to estimate the C_{lim}, such that $1 - \frac{(C_{req} - C_{lim})}{C_{req}} = QoS_{req}$. Based on this equation, we can derive C_{lim} that meets the QoS_{req}.

C. Consolidation Based On Performance Scalability

As discussed earlier, consolidating applications that do not scale linearly improves the energy efficiency by improving the overall utilization of the system, on the other hand consolidating scaling applications do not bring significant improvements as the system is already utilized by the application. Therefore, we run VMs that have $C_{req} > 11 * CoreFreq$ alone, as those VMs already utilize the hardware resources for a system with 12 cores. In order to choose the VMs to consolidate together, we rank the VMs according to their C_{req} values and pair the VMs starting from the bottom of the list, with the constraint $\sum_{i=1}^{n} C_{req_i} < R_{cap}$. As running more than 2 VMs for a 12-core system imposes more than 50% performance degradation, we evaluate our technique for the case where we consolidate 2 VMs at a time. However, the same algorithm can be applied to a system with more than 2 SMP VMs by prioritizing the VMs that have poor performance scalability.

In order to further improve the energy efficiency, we propose to distribute the CPU resources to VMs by prioritizing the ones that reaches to its maximum QoS with the smallest amount of CPU resources (i.e., poor scaling VMs). Therefore, the aggregate QoS of the system can be maximized by allocating the rest of the CPU resources to the co-runner VM. We set C_{lim}=C_{req} for the VMs that have lower C_{req} than its co-runner and allocate the rest of the CPU resources (i.e., $R_{cap} - C_{lim}$) to the co-runner VM which has a higher C_{req}.

VI. Experimental Results

We implement the **vCap** on the AMD Magny Cours processor based server. We deploy a management node to perform the administrative tasks on VMs. The management node collects runtime performance statistics from the ESXi hypervisor and power readings from the power meter. QoS guarantees and the power constraints are implemented as user-defined input parameters to the runtime routine. We keep the track of power estimation errors and the runtime routine continue to adjust the CPU resource limits by recalculating the R_{cap}, until the tracking error is smaller than 2W. QoS and R_{cap} estimation modules are implemented as Pyhton modules on the management node. In order to adjust the CPU resource limits, we create Perl modules using the vSphere SDK for Perl. The Perl module communicates with the ESXi and reconfigures the VMs CPU resource limits based on the input from the estimation modules in every 2 seconds, which is equal to the minimum sampling rate of the performance monitoring tool (i.e., `esxtop`). To implement the *GuestPacking* technique, we use the default `taskset` tool at the guest OS level to pack the threads onto smaller number of vCPUs. As **vCap** modules are implemented on a separate management node, the overhead of running **vCap** with 2s intervals is negligible.

In order to evaluate our technique, we randomly generate 10 workload sets, each of which consists of 10 applications selected among PARSEC and Cloudsuite applications. In Figure 7, we evaluate both the overall QoS of the system, as well as the QoS/watt metric to measure the energy efficiency. We compare our technique with previously proposed consolidation techniques that are based on CPU utilization and memory access per cycle (MPC) metrics. For CPU utilization and MPC based policies, we first rank the applications according to the selected metric. We then consolidate the highest ranked

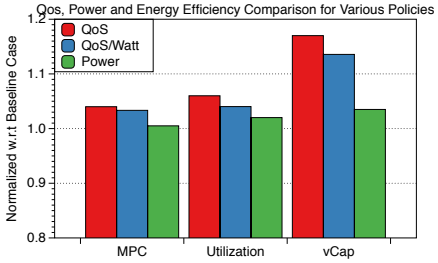

Fig. 7. Comparison of QoS, QoS/watt and power consumption of the server with various consolidation techniques. The proposed technique improves the overall QoS by 17% when compared to the baseline case, where VMs have no CPU resource limitations.

VM with the lowest ranked one and progress through the list and allocate equal resources to each VM. We normalize QoS, QoS/watt and power valus with respect to the baseline case, where we pair the VMs randomly and do not impose any CPU resource limits. **vCap** improves the QoS by 11% on average in comparison to the best performing previous policy. The energy efficiency of the server node also increases by 12% in comparison to the most energy-efficient previous policy.

We also test our technique under dynamically changing power caps to evaluate the power cap tracking accuracy. We change the power caps in every 8 seconds between 100W and 150W, similar to the real-life power regulation signals [2]. We compare the overall QoS for the proposed technique and the baseline case. **vCap** is able to adhere to the power cap 92% of the time within the $\pm$2W error margin, and 98% of the time within the $\pm$5W error margin.

Figure 8 shows the runtime behavior of *vCap* for the VM pair that is running `blackscholes` and `bodytrack`. We test our technique under the minimum QoS requirement of 60% for VM0 (i.e., `blackscholes`) and dynamically change the power caps between 125W and 160W. *vCap* accurately tracks the power cap accurately and satisfies the minimum QoS requirement that is required for VM0 by adaptively adjusting the resources in case of a QoS violation (e.g., t=6, t=22).

VII. Conclusions

Energy-related costs are among the most significant contributors to the total cost of ownership of the data centers. Thus, constraining the peak power consumption has become a common practice for cost management and reliable power delivery. As more than 50% of the cloud resources are virtualized, it becomes essential to design power capping solutions for virtualized servers. In tandem, multi-threaded applications start to emerge on the cloud resources from various application domains. Multi-threaded applications introduce additional

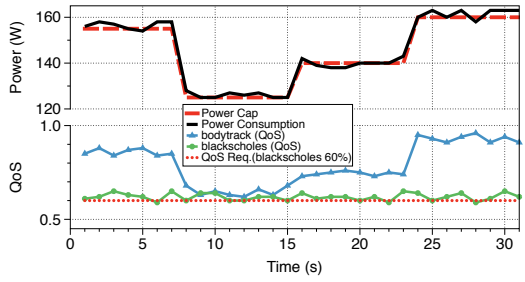

Fig. 8. Runtime behavior of **vCap** under power caps and QoS constraints for the VM group running `blackscholes` and `bodytrack`. *vCap* adheres to the power cap and ensures that the QoS guarantees are met.

challenges due to their more complex characteristics such as performance scalability. In this paper, we have proposed, **vCap**, a power capping technique for virtualized multi-core servers that improves the energy-efficiency of the server node by taking the applications characteristics into account. **vCap** identifies the VMs that exhibits poor performance scalability and consolidates them together. At runtime, **vCap** first estimates the total amount of CPU resources that meet the power caps. **vCap** then distributes the CPU resources among VMs according to the performance scalability of the VMs. We implemented vCap on a real-life multi-core server and show that vCap provides 12% higher energy efficiency in comparison to the state-of-the-art policies.

References

[1] R. Nathuji and K. Schwan, "VPM Tokens: Virtual Machine-aware Power Budgeting in Datacenters," in *International symposium on High Performance Distributed Computing (HPDC)*, 2008, pp. 119–128.

[2] I. Paschalidis, B. Li, and M. Caramanis, "A Market-Based Mechanism for Providing Demand-Side Regulation Service Reserves," in *Decision and Control and European Control Conference*, 2011, pp. 21 –26.

[3] L. Borovick, "The Benefits of a Virtualized Approach to Advanced-Level Network Services," *International Data Corporation (IDC), Whitepaper*, February 2011.

[4] G. Dhiman, G. Marchetti, and T. Rosing, "vGreen: A System for Energy-efficient Computing in Virtualized Environments," in *ISLPED*, 2009, pp. 243–248.

[5] C. Bienia, S. Kumar, J. P. Singh, and K. Li, "The PARSEC Benchmark Suite: Characterization and Architectural Implications," in *International Conference on Parallel Architectures and Compilation Techniques (PACT)*, October 2008.

[6] M. Ferdman and et al., "Clearing the clouds: a study of emerging scale-out workloads on modern hardware," in *International conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS)*, 2012, pp. 37–48.

[7] J. Li and J. Martinez, "Dynamic Power-performance Adaptation of Parallel Computation on Chip Multiprocessors," in *International Symposium on High-Performance Computer Architecture*, 2006, pp. 77–87.

[8] T. Samson, "AMD Brings Power Capping to New 45nm Opteron Line," http://www.infoworld.com/d/green-it/amd-brings-power-capping-new-45nm-opteron-line-906, 2009.

[9] H. David, E. Gorbatov, U. R. Hanebutte, R. Khanna, and C. Le, "RAPL: Memory Power Estimation and Capping," in *International symposium on Low power electronics and design (ISLPED)*, 2010, pp. 189–194.

[10] R. Raghavendra, P. Ranganathan, V. Talwar, Z. Wang, and X. Zhu, "No Power Struggles: Coordinated Multi-level Power Management for the Data Center," in *ASPLOS*, 2008, pp. 48–59.

[11] S. Reda, R. Cochran, and A. Coskun, "Adaptive power capping for servers with multithreaded workloads," *Micro, IEEE*, vol. 32, no. 5, pp. 64–75, 2012.

[12] K. Ma and X. Wang, "PGCapping: Exploiting Power Gating for Power Capping and Core Lifetime Balancing in CMPs," in *PACT*, 2012, pp. 13–22.

[13] I. Hwang, T. Kam, and M. Pedram, "A Study of the Effectiveness of CPU Consolidation in a Virtualized Multi-core Server System," in *ISLPED*, 2012, pp. 339–344.

[14] N. Vasić, D. Novaković, S. Miučin, D. Kostić, and R. Bianchini, "DejaVu: Accelerating Resource Allocation In Virtualized Environments," in *ASPLOS*, 2012, pp. 423–436.

[15] C. Delimitrou and C. Kozyrakis, "Paragon: Qos-aware scheduling for heterogeneous datacenters," ser. ASPLOS, 2013, pp. 77–88.

[16] J. Kim, M. Ruggiero, D. Atienza, and M. Lederberger, "Correlation-aware virtual machine allocation for energy-efficient datacenters," in *Proceedings of the Conference on Design, Automation and Test in Europe*, ser. DATE, 2013, pp. 1345–1350.

[17] 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.