VIP: Virtual Performance-State for Efficient Power Management of Virtual Machines

Ki-Dong Kang
DGIST
kd_kang@dgist.ac.kr

Mohammad Alian
University of Illinois, Urbana-Champaign
malian2@illinois.edu

Daehoon Kim
DGIST
dkim@dgist.ac.kr

Jaehyuk Huh
KAIST
jhhuh@kaist.ac.kr

Nam Sung Kim
University of Illinois, Urbana-Champaign
nskim@illinois.edu

ABSTRACT

A power management policy aims to improve energy efficiency by choosing an appropriate performance (voltage/frequency) state for a given core. In current virtualized environments, multiple virtual machines (VMs) running on the same core must follow a single power management policy governed by the hypervisor. However, we observe that such a per-core power management policy has two limitations. First, it cannot offer the flexibility of choosing a desirable power management policy for each VM (or client). Second, it often hurts the power efficiency of some or even all VMs especially when the VMs desire conflicting power management policies. To tackle these limitations, we propose a per-VM power management mechanism, `VIP` supporting **VI**rtual **P**erformance-state for each VM. Specifically, for VMs sharing a core, `VIP` allows each VM's guest OS to deploy its own desired power management policy while preventing such VMs from interfering/influencing each other's power management policy. That is, `VIP` can also facilitate a pricing model based on the choice of a power management policy. Second, identifying some inefficiency in strictly enforcing per-VM power management policies, we propose hypervisor-assisted techniques to further improve power and energy efficiency without compromising the key benefits of per-VM power management. To demonstrate the efficacy of `VIP`, we take a case that some VMs run CPU-intensive applications and other VMs run latency-sensitive applications sharing the same cores. Our evaluation shows that `VIP` reduces the overall energy consumption and improves the execution time of CPU-intensive applications compared with the default `ondemand` governor of Xen hypervisor up to 27% and 32%, respectively, without violating service level agreement (SLA) of latency-sensitive applications.

CCS CONCEPTS

• **Computer systems organization** → **Cloud computing**; *Heterogeneous (hybrid) systems*; Client-server architectures; • **Hardware** → *Enterprise level and data centers power issues*;

KEYWORDS

Virtualization, Power Management, Dynamic Voltage and Frequency Scaling, Cloud Computing

ACM Reference Format:
Ki-Dong Kang, Mohammad Alian, Daehoon Kim, Jaehyuk Huh, and Nam Sung Kim. 2018. VIP: Virtual Performance-State for Efficient Power Management of Virtual Machines. In *Proceedings of ACM Symposium on Cloud Computing , Carlsbad, CA, USA, October 11–13, 2018 (SoCC'18),* 12 pages. https://doi.org/10.1145/3267809.3267816

1 INTRODUCTION

The cloud servers have routinely adopted machine virtualization for various reasons, one of which is to improve energy efficiency. Such virtualization notably improves energy efficiency not only through consolidating workloads but also through power management choosing appropriate performance (voltage/frequency) states for cores. Thus, current hypervisors such as Xen and KVM support static and dynamic power management policies dynamically setting a Voltage/Frequency (V/F) level, similar to ones deployed by the Linux. However, the current hypervisors can promote only a single PM policy (i.e., host governor) per physical core (i.e., per-core PM). This poses a unique challenge for VMs sharing a physical core and running applications with opposite runtime characteristics in a time-shared manner (i.e., heterogeneous VMs); note that the consolidation policy often encourages heterogeneous VMs to share a physical core, since such VMs use different resources in the system [23].

For example, consider a PM governor that dynamically adjusts the V/F based on the utilization of a physical core (e.g., `ondemand` governor). Such a governor often fails to determine the V/F appropriate for consolidated VMs because it uses the overall utilization of the physical core sampled over a substantial period instead of that of individual VMs; the minimum sampling period set by Linux and Xen is 10ms due to various overheads caused by changing V/F [31]. When heterogeneous VMs share a physical core, a CPU-intensive VM may run at low V/F influenced by a lesser CPU consuming VM that previously ran on the core, or vice versa. Furthermore, the overall utilization can be mixed up if the both VMs run on a physical core within a sampling interval of the ondemand governor. This leads to sub-optimal V/F (or power efficiency) for both VMs. Hence, we need a mechanism that can provide strong PM isolation between VMs (i.e., a PM governor per VM instead of a physical core).

In this paper, tackling these limitations of the current hypervisors supporting only a PM governors per physical core, we propose VIP that supports **VI**rtual **P**erformance-state enabling per-VM PM for power and energy efficiency while allowing VMs to deploy a PM policy as they desire when VMs are consolidated. Specifically, to allow each VM to deploy its own PM governor optimized for its own runtime characteristics, VIP first defines *virtual V/F states* (i.e., virtual P-states) per vCPU and expose them to the guest OS. This allows the guest OS of each VM to deploy its own PM governor (i.e., guest governor) and manage the VM's virtual P-state through the guest governor based on its runtime characteristics. To properly support the guest governor of each VM in consolidated VM environments, VIP reflects the virtual P-state set by the guest governors to physical cores whenever the guest governors attempt to update the current virtual P-state of running vCPUs or the running vCPUs on cores are switched to other vCPUs demanding a different virtual P-state. Consequently, VIP supports strong PM isolation between heterogeneous VMs while sharing physical cores. That can lead to facilitating more diverse pricing models for the public cloud providers, since it allows each client to pick his/her own PM policy.

However, supporting such strict PM isolation is challenging when I/O-intensive VMs co-run with other VMs since the I/O-intensive VMs usually incur frequent vCPU switches on physical cores due to scheduling optimizations for improving I/O performance, such as boosting mechanism offered by Xen hypervisor. When demanding virtual P-states are different between I/O-intensive VMs and other VMs while sharing physical cores, the strict PM isolation leads to frequent P-state transitions.Consequently, the strict PM isolation may significantly degrade the overall performance and energy efficiency by frequent V/F switchings.

After identifying inefficiency of strictly enforcing per-VM PM policies, we propose hypervisor-assisted techniques that selectively enforce virtual P-states of I/O-intensive VMs to physical cores. Since I/O-intensive VMs usually run on cores for a short time and are blocked again, VIP enforces the virtual P-state of the I/O-intensive VMs to cores only when they are scheduled on idle cores. Moreover, to mitigate the negative impact on performance and energy efficiency by not enforcing virtual P-states, VIP lazily enforces virtual P-states when the priority of the boosted vCPUs are demoted to normal priority after continuously consuming a core for a certain amount of time. Therefore, VIP efficiently supports heterogeneous PM policies while mostly preserving each guest governor's policy.

We demonstrate the effectiveness of VIP when heterogeneous VMs, each of which runs a latency-critical On-Line Data-Intensive application (OLDI) and a CPU-intensive application, are consolidated while sharing the same cores. Supporting two independent guest ondemand governors for each VM, VIP reduces the overall energy consumption up to 27% while improving performance of CPU-intensive VMs up to 32% compared with the host ondemand governor. As heterogeneity of consolidated VMs increases, VIP further improves performance and energy efficiency of the consolidated VMs.

The rest of this paper is organized as follows. Section 2 and Section 3 explains the background and experimental methodology, respectively. Section 4 investigates the limitations of the PM of a current Xen hypervisor in consolidated VM environments. Section 5 proposes the overall architecture of VIP and discusses issues. Section 6 evaluates VIP. Section 7 describes the related work. Section 8 concludes this paper.

2 BACKGROUND

2.1 Hypervisor Power Management

The current Xen hypervisor provides four PM governors, ondemand, performance, powersave, and userspace, which are similar to the ones offered by the Linux OS. The performance, powersave, and userspace governors provide static PM policies while the ondemand governor adjusts the power states dynamically. The performance governor always operates cores at the highest V/F state for high performance while the powersave governor operates cores at the lowest V/F state for saving power and energy consumptions. Consequently, the performance governor shows high energy efficiency when cores are intensively utilized while the powersave governor is energy efficient when the core utilization is continuously low. Besides, the userspace governor allows a user to set the V/F of cores statically. In contrast, the ondemand [26] governor periodically adapts the V/F of cores based on the utilization of cores observed during the past interval; the default interval of Xen hypervisor's ondemand governor is 20ms. Hence, the ondemand governor can be more suitable when the utilization of cores varies over time.

Comparable to the aforementioned Xen's governors, VMware vSphere [6] supports four PM policies, balanced, high-performance, low-power, and custom. The balanced policy is the default one that adapts the operating frequency to reduce energy consumption while minimizing performance degradation, as the ondemand governor does. The high-performance policy does not decrease the operating frequency unless there is an explicit BIOS setting while the low-power policy aggressively reduces the operating frequency to reduce power consumption. The vSphere also provides the custom policy configurable by users. Meanwhile, other hosted hypervisors, such as Oracle VirtualBox, VMware workstation, and KVM, use or follow the PM policies offered by the host OSes.

A common limitation of the current hypervisor PM policies is the lack of any consideration for individual VMs. They can only set a global policy for all VMs. Even if they adjust the V/F dynamically based the CPU load, the CPU load is measured for each core potentially time-shared by multiple VMs. The measured CPU load is affected by the CPU usages of different VMs. We demonstrate the limitation quantitatively in Section 4.

2.2 Per-core DVFS

Although hypervisors and OSes allow each core to deploy a different PM governor, many commercial processors have only a single voltage domain. Thus, they support only chip-wide DVFS that maintains the same operating V/F for all the cores. However, such chip-wide DVFS may lead to energy inefficiency when heterogeneous applications run on different physical cores simultaneously. For example,

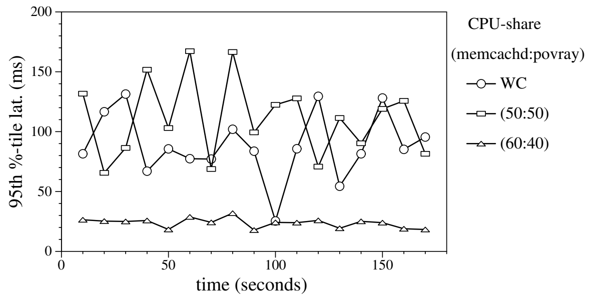


Figure 1: 95th percentile latency of `memcached` sharing a core with `povray` as CPU-share changes.

if an application that heavily utilizes a physical core increases the core frequency, other applications on different cores that may not intensively utilize cores also run at the same frequency. Meanwhile, AMD FX series processors support per-cluster Dynamic Frequency Scaling (DFS) without voltage scaling. Each cluster consists of two cores, thus the two cores in the same cluster share the frequency while all clusters operate at the same voltage. However, such per-cluster DFS technique still cannot effectively support per-core based PM governors of OSes and hypervisors.

To overcome such limitation of techniques with the single voltage domain, the processor industry began to support per-core DVFS with multi-voltage domains for high-end server processors. For example, recent Intel Xeon processors (e.g., E5 v4 series and Scalable series processors) support per-core DVFS. The processors supporting per-core DVFS have potentials that can effectively improve energy efficiency with per-core PM governors offered by OSes and hypervisors when the heterogeneous applications run on different cores in the same processor package.

2.3 Scheduling and Performance Isolation

The Xen hypervisor uses a credit-based scheduler by default that attempts to schedule virtual CPUs (vCPUs) sharing physical cores in a time-shared manner. The credit scheduler allocates the `weight` representing a relative CPU share to each VM. By default, the Xen hypervisor allocates the same `weight` to each VM for fairness in the CPU share; the hypervisor can also manipulate the CPU share of consolidated VMs by adjusting the `weight`. The credit-based scheduler allocates the `credit` to vCPUs based on its `weight`, and deducts the `credit` as much as the vCPU runs on physical cores. To maximize CPU utilization, the credit scheduler schedules vCPUs in work-conserving (WC) manner that schedules vCPUs that already ran out of its allocated `credits`, if no runnable vCPU is in a run queue.

However, the credit-based WC scheduler can be undesirable for utility-based cloud computing environments running latency-critical, OLDI applications that need to satisfy a specific Service Level Agreement (SLA), as it entails poor performance isolation between VMs and thus increases the tail latency of services. With the WC scheduling, the actual CPU allocation for VMs fluctuates significantly, as the VM consolidation ratios or loads of VMs change. In particular, for latency-sensitive applications in client-server architecture, the WC scheduling often leads to violation of SLA by increasing high

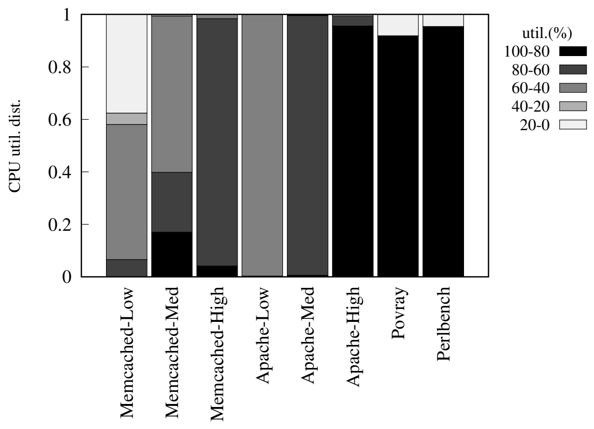


Figure 2: CPU-utilization with `ondemand` governor.

percentile latency since the latency-sensitive applications cannot reserve enough CPU time due to the fluctuated CPU loads.

To prevent a VM from disrupting the consistent performance of other VMs, the `capping` mechanism can limit the maximum CPU utilization of each VM. By adjusting `cap` values, the hypervisor can differentiate the CPU share of consolidated VMs without interference by other consolidated VMs.For public clouds where the performance consistency and QoS of each VM is often more important than the overall system utilization, guaranteeing a fixed CPU resource to a VM consistently is common as exemplified by Amazon EC2 [3].

Figure 1 shows the 95th percentile latency of `memcached` co-running with `povray` intensively consuming CPU with `capping` mechanism compared with WC scheduling. The `WC` scheduling fails to maintain consistent 95th percentile latency due to the interference of the co-running VM running the CPU-intensive batch application. While the equal allocation (i.e., 50:50) still shows the heavy fluctuation of 95th percentile latency, more CPU-shares for `memcached` entail fairly low and constant 95th percentile latency. Consequently, such strong performance isolation for the latency-critical workloads is essential in consolidated environments.

2.4 CPU Utilization and P-states

For high energy-efficiency, the dominant factor in choosing a proper V/F(voltage/frequency) state (i.e., P-state) is *CPU utilization*. The P-state with the highest V/F (i.e., P0) is likely to provide the best performance for CPU-intensive applications that highly utilize physical cores. On the contrary, PM governors do not need to set the highest V/F when the core utilization is low. Unsuitable V/F state for runtime behaviors of applications may not only degrade performance, but also unnecessarily increase power or energy consumptions without notable performance improvement. Consequently, the current default governor of Xen hypervisor (i.e., `ondemand` governor) dynamically adjusts the V/F of a physical core based on CPU loads.

Figure 2 plots the distribution of CPU-utilization obtained every 20ms (i.e., the default DVFS sampling period of Xen hypervisor) during the whole execution time of benchmarks. We run two CPU-intensive benchmarks, `povray` and `perlbench` from

Table 1: OLDI benchmark configuration.

benchmark	load level	configuration
memcached	low	5K pps
	medium	14K pps
	high	50K pps
apache	low	concurrency 1
	medium	concurrency 2
	high	concurrency 21

Table 2: Benchmark with abbreviation.

abbrv.	benchmark
PO	povray
PE	perlbench
ML	memcached low
MM	memcached medium
MH	memcached high
AL	apache low
AM	apache medium
AH	apache high

SPECCPU-2006 [5], and two OLDI benchmarks, `memcached` [14] and `apache` [1] with three different load levels, `low`, `medium`, and `high`). We run benchmarks on a single VM on the Xen hypervisor [10] with the default `ondemand` governor.

Figure 2 presents completely different CPU utilizations among benchmarks and load-levels. `povray` and `perlbench` mostly utilize a physical core more than 90%, making the core mostly runs at the highest V/F (i.e., P0) with the `ondemand` governor. Whereas, `memcached` and `apache` show the different CPU-utilizations according to the load-level. As the load-level increases, the both benchmarks show the more CPU utilizations while showing slightly different distributions. `apache` shows relatively constant distributions while `memcached` shows more various distributions. This is because `apache` uniformly generates network packets while `memcached` generates a burst of packets periodically.

3 EXPERIMENTAL METHODOLOGY

We investigate the limitation of current hypervisor's PM in consolidated VM environments and evaluate our mechanisms in Section 4 and Section 6, respectively. We run the Xen hypervisor [10] on an eight-core Intel Xeon Silver 4110 Processor supporting per-core V/F (i.e., P-state). Each core supports 14 different frequencies ranging from 0.8Ghz to 2.1Ghz. We evaluate the dynamic `ondemand` governor offered by the Xen hypervisor using the default configuration of 20ms sampling period and 80% threshold. As a guest OS, we use Linux OS also offering the `ondemand` governor.

For consolidated VM environments, we run eight heterogeneous single-core VMs on a separated CPU pool consisting of four physical cores using `cpupool` mechanism offered by Xen hypervisor. We pair a CPU-intensive benchmark and an On-line Data-Intensive(OLDI) benchmark on each core. As we did for Figure 2, we use two CPU-intensive benchmarks, `povray` and `perlbench`, and two OLDI benchmarks, `memcached` and `apache` with three different load levels described in Table 1. We find the maximum number of packets per second (pps) not violating SLA and use the loads as the high-load level for `memcached`. We also find the maximum number of requests concurrently generated by the client (i.e., concurrency) not violating SLA and use the concurrency level as the high-load level for `apache`. Besides, we use lower loads (i.e., medium- and low-load levels) showing lower CPU utilization for various scenarios.

We do evaluations with `capped` scheduling offering strong performance isolation among consolidated VMs; such non-work conserving scheduling is common in the cloud environments where performance consistency and QoS of each VM is critical (e.g., Amazon EC2) [3, 9]. Through the performance isolation, the OLDI VMs and CPU-intensive VMs reserve the CPU share by 60% and 40%, respectively. We measure the actual energy consumption by reading model specific registers (MSRs) maintaining consumed energy by the processor package offered by Intel processors.

4 LIMITATION OF CURRENT PER-CORE BASED POWER MANAGEMENT

In consolidated VM environments, multiple VMs often run on a physical core in a time-shared manner. The current hypervisors, however, support only per-core PM, allowing one PM policy for co-running VMs sharing the core. Furthermore, dynamic PM governors such as `ondemand` governor only track the average core utilization of all the co-running VMs to adjust the V/F. Thus, when heterogeneous VMs run on a single physical core, some or all VMs may run at the sub-optimal V/F, affected by co-running VMs. It is inevitable that the energy efficiency of some VMs may be significantly degraded. In this section, we discuss the negative impact of the current hypervisor's per-core based PM when heterogeneous VMs co-run.

4.1 Static PM Governor

The static governors offered by the hypervisor are essentially not energy efficient with varying CPU loads since they statically set and do not adjust the V/F. The lower V/F statically set by the governor than the demanded performance by VMs continuously degrade their performance while the higher V/F unnecessarily consumes extra power. Moreover, sharing a core with multiple VMs exacerbates inefficiency since they must follow the same static PM policy even though they show heterogeneous runtime behaviors.

The per-core static PM governors cannot even offer constant performance for consolidated VMs when each core deploys a different governor. For example, a VM running on a core deploying `performance` governor shows the higher performance along with more power consumption than another VM runs on a core deploying `powersave` governor. Furthermore, a VM can be migrated to another core operating at a different V/F for load balancing among cores, leading to different performance and power consumption.

To support per-VM PM for single core VMs, we can pin VMs demanding the same P-state on the same core. For example, the scheduler pins VMs running CPU-intensive applications on a core deploying `performance` governor while pinning VMs running I/O-intensive applications on a core deploying `powersave` governor. However, such pinning for per-VM PM limits flexibility of VM

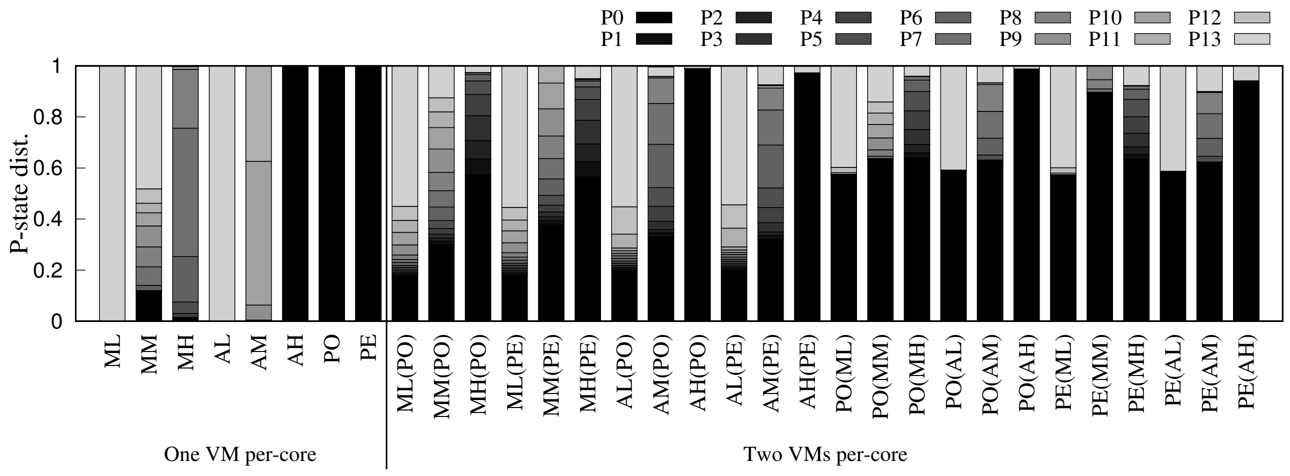


Figure 3: P-state distribution by `ondemand` governor

schedulings, degrading the overall CPU utilization and performance; multi-core VMs complicate the pinning for supporting per-VM PM.

Due to the limitations of static governors as mentioned earlier, per-core based static governors are not desirable for consolidated VM environments. The CPU loads of VMs running on the same core are usually fluctuated, moreover, the number of VMs running on the core changes. Hence, in this paper, we do not quantitatively evaluate the static governors offered by the hypervisor.

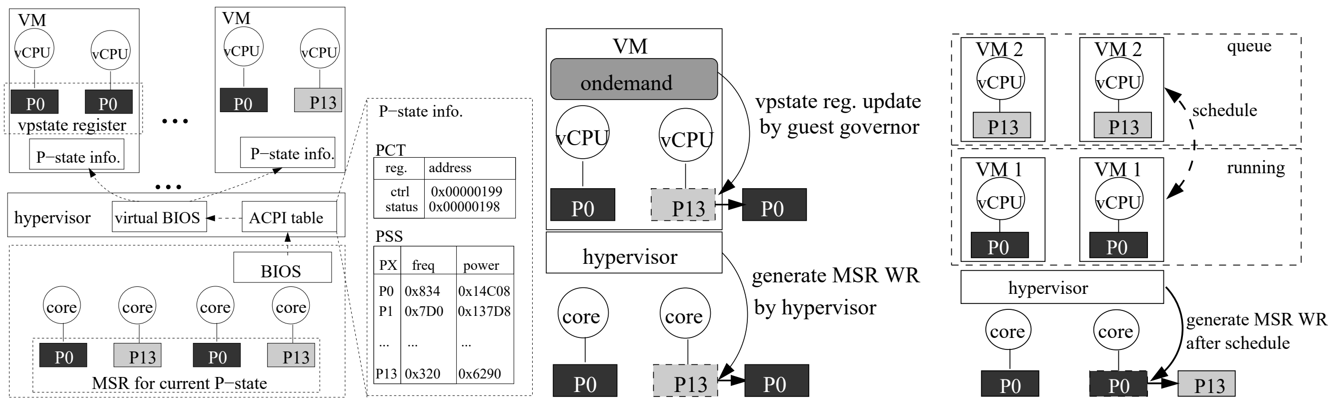
4.2 Dynamic PM Governor with Heterogeneous VMs

The `ondemand` governor is commonly used as a default governor since it attempts to maximize energy efficiency by dynamically adjusting the V/F of cores based on the utilization of physical cores at runtime. The `ondemand` governor lowers the operating V/F of a core running a VM with a low CPU utilization while raising the V/F with a high CPU utilization. However, we observe that the `ondemand` governor is not sufficiently effective when heterogeneous VMs run on the same core with strong performance isolation offered by `capped` scheduling.

In consolidated VM environments where heterogeneous VMs share cores with performance isolation, the `ondemand` governor can degrade performance of CPU-intensive VMs as well as fail to reduce the power consumption. Basically, the hypervisor schedules VMs based on the allocated time slice (i.e., credits) regardless of the invocation period of the `ondemand` governor. The scheduled VMs can run on cores until they ran out of the credits, unless they are scheduled out due to blocked I/O or idleness, or preempted by other vCPUs with the higher priority (e.g., `BOOST` priority of Xen hypervisor's credit scheduler). Consequently, multiple VMs inevitably time-share a core in the same invocation period of the `ondemand` governor (e.g., 20ms), mixing core utilization of the VMs. Such mixed core utilization leads to setting the sub-optimal V/F by the `ondemand` governor since it solely measures the overall utilization of a physical core in the last interval.

Figure 3 plots the runtime breakdown by P-states with the `ondemand` governor when one VM runs alone (Figure 3) and a pair of VMs on a single core, each of which runs a CPU-intensive and an OLDI benchmark with performance isolation through `capped` scheduling (Figure 3). In Figure 3, for OLDI benchmarks, the `ondemand` governor varies the P-state distributions of cores according to its load-level. As the load level increases, the VMs more run at the higher V/F. For example, the `ondemand` governor mostly sets the lowest V/F (i.e., P13) for `memcached` with the low-load level while it stays longer at higher V/F states for the medium- and high- load levels. As plotted in Figure 2, `memcached` shows more varied P-state distributions due to periodic packet bursts as discussed in Section 2.4. Meanwhile, the `ondemand` governor mostly maximizes the operating frequency for CPU-intensive VMs due to their high core utilization. As shown in Figure 3, when two heterogeneous VMs (i.e., a CPU-intensive VM and an OLDI VM) share a core with performance isolation, `ondemand` governor sets the sub-optimal P-state for the both VMs. As heterogeneity of consolidated VMs increases, the VMs run more at the inefficient P-state with the host governor from the Xen hypervisor. The CPU-intensive VMs running `povray` and `perlbench` spend a significant portion of execution times at lower V/F, potentially increasing the execution times. For example, while consolidating `povray` and `memcached` with the low-load level, `povray` runs at lower P-states than P0 for 42% of the execution time, degrading performance significantly; note that `povray` mostly runs at P0 while running alone with the `ondemand` governor. The OLDI VMs running `memcached` and `apache` are running at higher V/F unnecessarily, wasting power. `memcached` runs at higher P-states than P13 for 44% of the execution time while co-running with `povray`, consuming extra power; note that `memcached` with the low-load level mostly runs at P13 when it runs alone. This ineffective governor behavior is due to the core utilization sampling is done independently from VM context changes. `perlbench` and `apache` also run at inefficient P-states for a considerable amount of time due to co-running OLDI VMs showing different runtime behaviors.

As the load level of the OLDI VMs increases, the `ondemand` governor is likely to set the higher V/F also, increasing energy efficiency for the both CPU-intensive and OLDI VMs. Since the both



(a) Enable virtual P-states and *vpstate register*. **(b) Update by guest governor.** **(c) Update by scheduler.**

Figure 4: VIP: enabling and reflecting virtual P-states for guest governors.

CPU-intensive and OLDI VM with the high-load level show the high CPU utilization, the measured utilization is also high even though the utilization of the both VMs are mixed. This means current `ondemand` governor works well when the heterogeneity among consolidated VMs decreases. However, the `ondemand` governor can set higher V/F unnecessarily consuming power when the utilization of VMs, each of which shows low CPU utilization, are mixed.

Furthermore, the network packet bursts of `memcached` leads to unexpected results in P-state distributions when co-running with other VMs. `memcached` makes `perlbench` run more at P0 with the medium-load level than the high-load level, but it makes `povray` run more at P0 with the high-load level. This is because network packet bursts unpredictably vary the overall CPU utilization according to scheduling behaviors even though `povray` and `perlbench` show the almost same P-state distribution while running alone.

5 VIP ARCHITECTURE

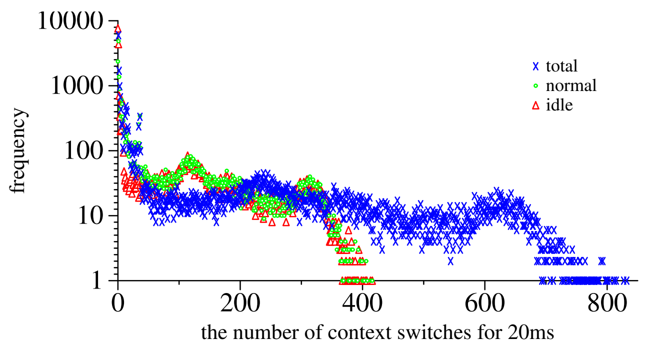
In this section, we describe `VIP` that supports **VI**rtual **P**erformance states for per-VM power management for consolidated VM environments. By enabling virtual V/F (i.e., virtual P-state) per vCPU, and enforcing the virtual P-state of the currently scheduled vCPUs to physical cores, `VIP` first allows each VM to promote its own PM governor (i.e., guest governor). The virtual P-state of the scheduled vCPU is reflected in a physical core when the guest governor attempts to change the virtual P-state or the vCPU is scheduled on the physical core at the beginning of a time slice. To tackle inefficiency of enforcing virtual P-state per schedule event, `VIP` also proposes hypervisor-assisted techniques that determine appropriate enforcement of virtual P-state of the guest governor based on the status of the scheduled vCPU.

5.1 Supporting Per-VM Power Management

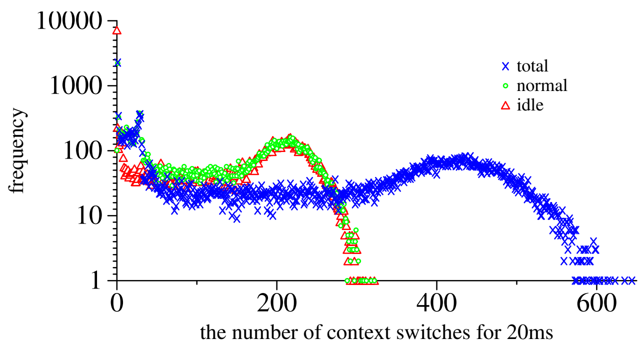
Enabling Virtual P-state. In non-virtualized systems, BIOS provides the OS with the PM information supported by physical cores with the `ACPI` (Advanced Configuration and Power Interface) tables [2]. The ACPI tables maintain two essential objects, `PCT` (Performance Control) and `PSS` (Performance Supported States). The `PCT` object describes two model specific registers (MSRs) to control and monitor the current P-state of each core. One register records the P-state recently requested by the governor (i.e., `control register`), and the other one stores the current P-state (i.e., `status register`). The values of both registers can be different for a moment after updating `control register` due to P-state transition latency. Besides, since the chip-wide or clustered DVFS maintains the same current P-state for all the cores in a chip or the same cluster, the two MSRs also may have different values. The `PSS` object describes the number of supported P-states, and other information regarding each P-state, such as core frequency, power, transition latency.

In current virtualized systems, such P-state information is accessible only by the hypervisor, but not by the guest OSes. BIOS exposes registers to control the current P-state, and available set of P-states to hypervisor through the objects (i.e., `PCT` and `PSS`) specified in the ACPI tables when the hypervisor is booted. By reading and changing the registers, the hypervisor allows the host PM governor to change the P-state of each core, as the OS does in non-virtualized systems. In contrast, the guest OS cannot manipulate the current P-states of physical cores at all. In the current implementation, when the guest OS is booted, although the virtual BIOS initiated by `hvmloader` also loads the ACPI tables for a guest OS, it does not provide any necessary objects including `PCT` and `PSS` to the guest OS.

To allow the guest OS to manipulate the P-states, `VIP` implements virtual P-states per vCPU. Specifically, modifying the virtual BIOS, `VIP` exposes the necessary objects including the P-state information of physical cores (i.e., `PCT` and `PSS`) to the guest OS when booting a VM. That is, `VIP` can offer the same set of available P-states as physical cores for each vCPU of VMs. For example, if the physical cores support 14 P-states (P0–P13), a guest governor can choose a current virtual P-state amongst those 14 P-states for a vCPU. Figure 4a illustrates how `VIP` enables virtual P-states for VMs. To maintain a virtual P-state per core, `VIP` defines a `vpstate register`, which can be controlled by the guest governor, as depicted in Figure 4a. As a native OS or hypervisor does, the guest



(a) memcached low.



(b) memcached high.

Figure 5: The number of vCPU context switches on a physical core.

governor uses the `vpstate register` to check and change the virtual P-state of a vCPU in `VIP`.

Enforcing Virtual P-state. In addition to enabling virtual P-states, to deploy a guest governor, the hypervisor must adequately reflect the current virtual P-state manipulated by the guest governor to a physical core. Figure 4b and Figure 4c illustrate two update scenarios of the current P-state with the `VIP` architecture, assuming a VM with two vCPUs running on two physical cores supporting 14 P-states (i.e., P0 – P13).

Figure 4b describes an example of enforcing a virtual P-state update initiated by the guest `ondemand` governor.In this example, the guest `ondemand` governor desires the P0 (highest V/F) state for the second vCPU, as the vCPU is highly utilized in the last sampling period. Subsequently, the guest `ondemand` governor updates its `vpstate register` to change the virtual P-state of the vCPU to P0. As soon as detecting the update of the `vpstate register` by the guest governor, the hypervisor attempts to update the `control register` of the physical core to change the P-state of the physical core where the vCPU is scheduled. Such a P-state update is conducted at every sampling period set by the guest `ondemand` governor whenever it decides to change its current P-state for the next sampling period.

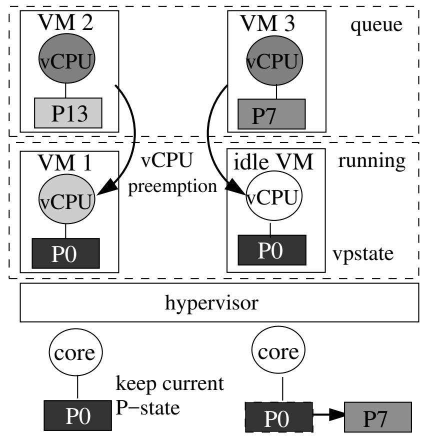
In a consolidated VM environment, vCPUs with their own virtual P-states are scheduled in and out over time. Therefore, when a vCPU is scheduled on a core, the hypervisor also needs to enforce the virtual P-state of the vCPU to the scheduled physical core at the beginning of its time slice. Figure 4c depicts an example of such a case. In this example, two VMs (denoted by VM1 and VM2), each of which uses two vCPUs, deploy the `performance` and `powersave` governors, respectively. That is, the guest governor of VM1 and VM2 sets the P-state of both the vCPUs to P0 (highest V/F) and P13 (lowest V/F), respectively. When scheduling in a vCPU of VM2 deploying the `powersave` governor to a physical core (e.g., the second physical core in Figure 4c), the hypervisor checks the `vpstate register` of the vCPU and updates the `control register` of the physical core to change the core's P-state from P0 previously set by the `performance` governor of VM1 to P13. Note that the `ondemand` host governor of traditional hypervisors periodically updates the physical P-state without considering VM scheduling.

5.2 Hypervisor-assisted Per-VM Power Management

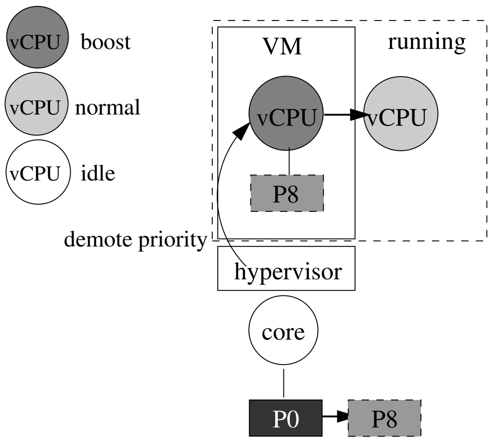
Strictly enforcing virtual P-states driven by guest governors whenever vCPUs are scheduled on physical cores may cause significant delays and energy inefficiency due to frequent P-state transitions when vCPUs are frequently changed on cores. For improving I/O performance, hypervisor's scheduler usually promotes the priority of vCPUs of I/O-intensive VMs (e.g., `BOOST` priority of Xen hypervisor's credit scheduler). The promoted vCPUs preempt currently running vCPUs, and then, run on cores. However, such vCPUs of the I/O-intensive VMs usually run on the cores for a very short time, and then, are scheduled out again, leading to frequent vCPU switches by the scheduler. Since the strict enforcement incurs P-state transitions while switching vCPUs on cores when they have a different virtual P-state, frequent vCPU switches may lead to frequent P-state transitions. Thus, the strict enforcement incurring frequent P-state transitions is not an efficient mechanism due to delays and power consumptions by P-state transitions.

Furthermore, the P-state transition can even be ignored when OSes or hypervisors overwrite a `control register` for updating a current P-state of a core before the P-state previously recorded in the `control register` is reflected in the core; the `acpi-cpufreq` driver in Linux OS defines the minimum P-state re-transition delay (i.e., 10ms) by multiplying a transition latency (i.e., 10μs in Intel Xeon processors) by 1000. However, we observe that the re-transitions within less than 10ms also work by modifying the driver, but the former P-state update is mostly ignored when a successive P-state update occurs within 1ms in Intel Xeon processors.

Figure 5 counts the number of scheduling events in each sampling period (i.e., 20ms) of the `ondemand` governor when `memcached` co-runs with `povray`. We also separately counts the number of schedulings when a vCPU preempts a normal vCPU and runs after an idle vCPU, denoted as `normal` and `idle`, respectively. We observe many schedulings more than 800 times in a single sampling interval of the `ondemand` governor with `memcached` with the low-load level; `memcached` with the high-load level also shows schedulings more than 600 times. Amongst the total scheduling events, almost half of scheduling events are from `normal`, leading



(a) Selectively enforcing virtual P-states of scheduled vCPUs based on previously running vCPUs on cores.



(b) Lazily enforcing virtual P-states when demoting scheduling priority of vCPUs.

Figure 6: Hypervisor-assisted power management.

to P-state transitions. This means P-state transitions more than 400 times can occur in 20ms with strictly enforcing virtual P-states. Thus, strictly enforcing virtual P-state is not feasible with I/O-intensive VMs.

Tackling the aforementioned limitations of the strict virtual P-state enforcement, we propose hypervisor-assisted techniques that avoid inefficient enforcement of virtual P-states considering the scheduling status of vCPUs. To avoid frequent P-state transitions while co-running with I/O-intensive VMs, `VIP` selectively enforces virtual P-states of scheduled vCPUs with the high priority (e.g., `BOOST` in Xen's credit scheduler) on the basis of previously running vCPUs. `VIP` enforces the virtual P-states of the boosted vCPUs only when they run after idle vCPUs while not enforcing when they preempt other normal vCPUs as described in Figure 6a. This is because, switches between normal vCPUs with different virtual P-states incur P-state transitions while other vCPUs that may incur P-state transitions are unlikely to be scheduled sooner when cores are idle. In Figure 6a, `VIP` enforces the virtual P-state of VM3 while not enforcing the virtual P-state of VM2 since VM2 preempts a vCPU of VM1. Consequently, `VIP` avoids the frequent P-state transitions while co-running with I/O-intensive VMs. Furthermore, not enforcing virtual P-states for the boosted vCPUs does not notably impact on the overall performance and power in most cases since they are usually scheduled out in a short time.

However, the boosted vCPUs occasionally consume cores for a long time when they run CPU-intensive and I/O-intensive applications on the vCPU concurrently, or the running application generates a lot of I/O requests while consuming the CPU. Thus, the boosted vCPUs may run for a long time at the inefficient V/F set by another vCPU since they do not enforce virtual P-states after preempting other normal vCPUs. This significantly degrades performance when vCPUs runs at the lower V/F than their virtual P-state while unnecessarily consuming extra power when they run at the higher V/F.

To mitigate inefficiency by running at the inefficient P-state set by another vCPU, `VIP` lazily enforces virtual P-states for the boosted vCPUs in case they continuously run on the same core for a certain amount of time. The credit scheduler demotes the priority of the boosted vCPUs (e.g., `UNDER` state in Xen's credit scheduler) when they run for a certain amount of time, implying that the vCPUs are not needed to be boosted anymore since they sufficiently consume cores; the default is 10 ms with 30 ms time slice in Xen's credit scheduler. Along with the priority demotion, `VIP` enforces virtual P-states for the boosted vCPUs when the scheduler demotes the their priority. In Figure 6b, the virtual P-state of the boosted vCPU (i.e., P8) runs at P0 set by the previously running vCPU on the core. The P8 is enforced when the priority of vCPU is demoted to normal priority. With the default time slice of Xen hypervisor (i.e., 30 ms), the boosted vCPUs can run at inefficient P-states for 10 ms, and run for the other 20 ms at demanding P-states. Consequently, `VIP` improves energy efficiency of vCPUs by reducing running time on cores at inefficient P-states while avoiding frequent P-state transitions.

5.3 Discussion

Energy-based pricing model. Since energy consumption is the most significant aspect of the total cost of ownership (TCO), public cloud providers can adopt a pay-as-you-go pricing model for energy consumption through power metering schemes [17, 21], as they do for computing resources. With the energy-based pricing model, a tenant may choose a PM governor to reduce energy consumption regardless of application's characteristics. For example, the tenant can promote `powersave` governor for supposedly CPU-intensive VMs to reduce the charge for the VMs. The host dynamic PM governor based on runtime behaviors including `ondemand` governor cannot support such PM policies independent of the behaviors. However, `VIP` supports guest PM governors regardless of its policies through per-VM management.

Ondemand governor with a short interval. The traditional host `ondemand` governor can reduce the sampling/DVFS interval to capture fine-grained behaviors of VMs sharing a core. There are

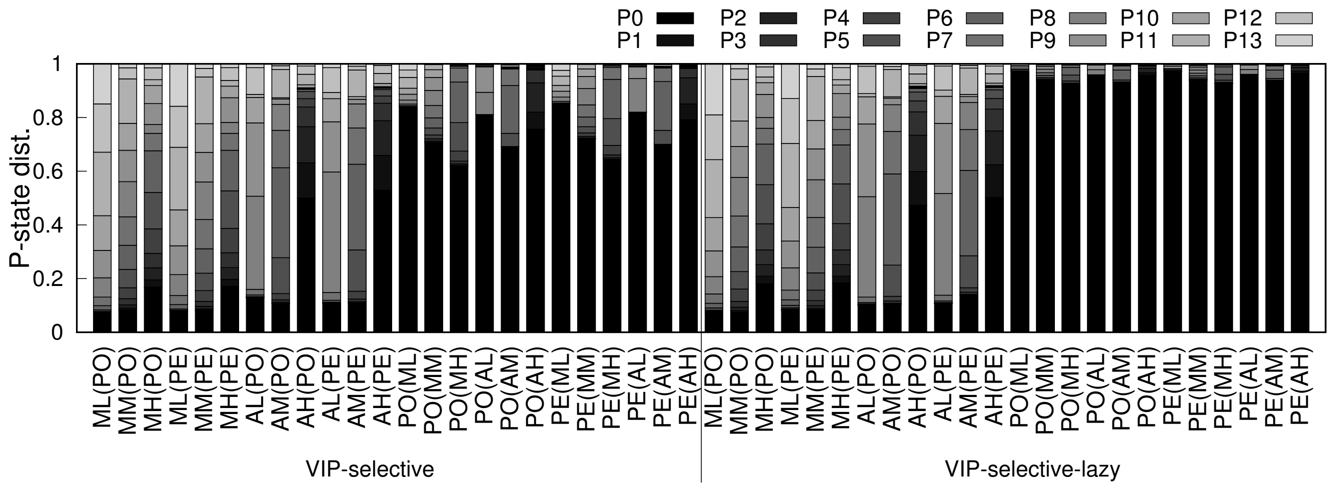


Figure 7: P-state distribution of `VIP`.

two performance overhead components in changing P-states: (1) changing voltage/frequency and (2) invoking the PM software-stack. In particular, we observe that the overhead effect of (2) becomes notable when we *force* sub-10ms sampling/DVFS intervals (e.g., 1ms) and the workload behavior keeps changing; 10ms is the minimum interval supported by Xen and Linux `ondemand` governors while the default sampling/DVFS interval of Xen's `ondemand` governor is 20ms considering such negative effects of very short intervals. Furthermore, governors invoked at very short intervals unnecessarily react to short-lived or transient changes in program behavior. This makes the governors keep making sub-optimal decisions over intervals [27]. For example, some OLDI benchmarks show the even worse response times with sub-10ms intervals [8]. Lastly, frequent invocations disturb entering a C-state when a core is mostly idle. This leads to unnecessary power consumption.

Highly consolidated environments. Although `VIP` supports an independent PM for each VM even in highly consolidated VM environments, assuming that all consolidated VMs deploy the dynamic governor (e.g., `ondemand` governor), the more number of VMs can lead to more overheads due to frequent P-state transitions. However, the consolidation ratio does not proportionally contribute to the number of P-state transitions because only currently running vCPUs on cores can attempt to update the P-state of the cores. Note that a vCPU can continuously run for 30ms on a core with the default configuration of Xen's credit scheduler; `VIP` also proposes hypervisor-assisted mechanisms for a case that vCPUs are frequently switched on cores. Furthermore, the guest `ondemand` governors do not always change their P-state every sampling period (e.g., 10ms). They update the P-state only when CPU utilization is varied.

6 EVALUATION

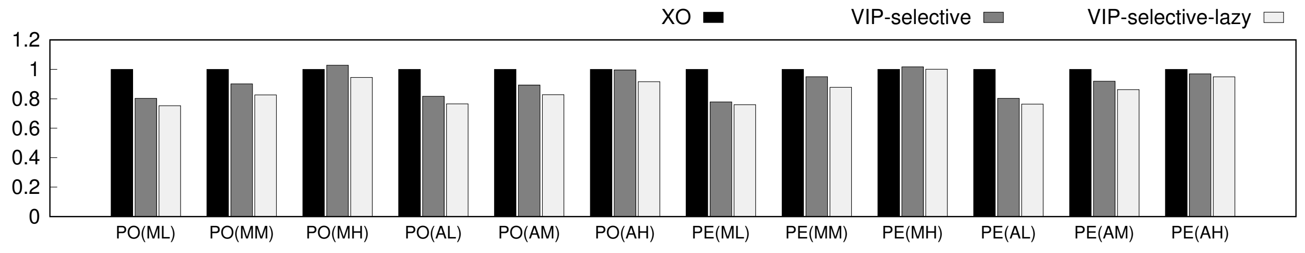
In this section, we show that `VIP` improves energy efficiency in consolidated VM environments. We compare `VIP` running a guest `ondemand` governor provided by the Linux OS in each guest VM against host `ondemand` governor provided by the Xen hypervisor, in terms of execution time and energy consumption. We run four CPU-intensive benchmarks and four `memcached` with eight single core VMs on four physical cores, while pairing heterogeneous VMs on a physical core. In addition, we run two CPU-intensive benchmarks and two `apache` with four single core VMs on two physical cores since four `apache` benchmarks saturate our 1Gb network bandwidth while not differentiating CPU loads according to load levels. We describe the detailed scenario in Section 3.

`VIP` allows each VM to deploy a static governor also, such as `performance`, `powersave`, and `userspace` governor. When the CPU utilization of VMs does not vary during the whole runtime, guest static governors also can present sufficient energy efficiency. Furthermore, since static governors provide constant performance with fixed P-states, it can be crucial to support the guest static governors in multi-tenant clouds. In this paper, we do not demonstrate `VIP` with static guest governors since there is no technical difference supporting guest `ondemand` governors and guest static governors except that the `ondemand` governor periodically adjusts virtual P-states.

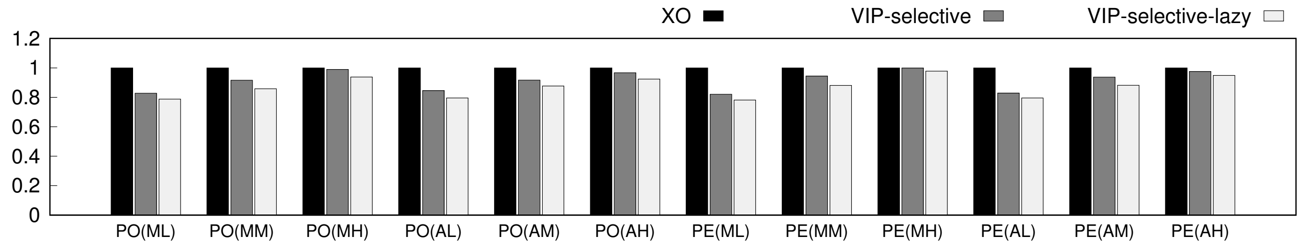
We evaluate two approaches, `VIP-selective` and `VIP-selective-lazy`. Since strictly enforcing virtual P-state is mostly ignored while co-running with OLDI VMs, we do not evaluate the strict version of `VIP`. `VIP-selective` and `VIP-selective-lazy` selectively enforce virtual P-states with high priority (i.e., `BOOST`) based on the previously running vCPU on a core. `VIP-selective-lazy` lazily enforces virtual P-states of boosted vCPUs when scheduler demotes their priority while selectively enforcing virtual P-states as `VIP-selective`. We denote the host `ondemand` governor from the Xen hypervisor as `XO`.

6.1 P-state Distribution

To demonstrate that `VIP` effectively preserves guest PM policies, we plot P-state distributions of `VIP-selective` and `VIP-selective-lazy` in Figure 7. For OLDI VMs, `VIP`



(a) Normalized execution time.



(b) Normalized energy consumption.

Figure 8: Comparison of Xen ondemand governor and VIP.

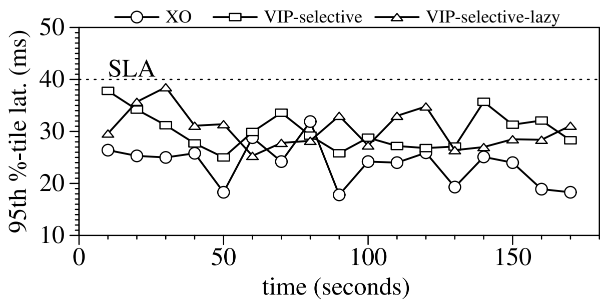


Figure 9: 95th percentile latency of memcached high configuration sharing a core.

fairly preserves the guest `ondemand` governor's policy by enforcing virtual P-states. `memcached` with the low-load level co-running with `povray` mostly runs at P-states with lower V/F with the both `VIP-selective` and `VIP-selective-lazy`; `VIP-selective` and `VIP-selective-lazy` present similar results for OLDI VMs since the boosted vCPUs rarely run for a long time after preempting other normal vCPUs. For `memcached`, the P-state distribution of co-run by the guest `ondemand` governor is slightly different with the one of solo-run by the hypervisor's `ondemand` governor since the measured utilizations of vCPUs by the guest governors are slightly increased while sharing cores with other VMs.

For CPU-intensive VMs, `VIP-selective-lazy` shows the almost same P-state distributions between solo-run and co-run while the CPU-intensive VMs run at lower P-states than P0 for some portion of the execution time with `VIP-selective`. With `VIP-selective`, `povray` co-running with `memcached` with the low-load level runs at P0 and lower P-states than P0, for 84% and 16% of the execution time, respectively. This is because the `VIP-selective` does not enforce virtual P-states of the boosted vCPUs of CPU-intensive VMs when they preempt the OLDI VMs running at low P-states; vCPUs of CPU-intensive VMs are occasionally boosted for handling hypervisor callback interrupts. In contrast, with `VIP-selective-lazy`, `povray` mostly runs at P0 since `VIP` lazily enforces the virtual P-state of `povray` in 10ms when the vCPU's priority is demoted.

6.2 Performance and Energy Consumption

Figure 8 plots the total energy consumption of consolidated VMs and execution time of CPU-intensive VMs. The results of `VIP-selective` and `VIP-selective-lazy` are normalized to ones of `XO`. For all pairs, `VIP-selective-lazy` shows the best performance and energy consumption. `VIP-selective-lazy` improves the execution time of CPU-intensive VMs and the total energy consumption up to 32% and 26%, respectively, when the heterogeneity of consolidated VMs is maximized by consolidating the CPU-intensive VMs and `memcached` with the low-load level. Similarly, with `apache`, `VIP-selective-lazy` improves the execution time of the co-running CPU-intensive VMs up to 31% and reduces energy consumption up to 27% with the low-load level.

As plotted, as the heterogeneity of consolidated VMs increases, `VIP` shows further improvements compared with `XO`. This is because the heterogeneity makes per-core based `ondemand` governor set more inappropriate P-states even for the all VMs sharing a core.

As the heterogeneity decreases, VIP shows less improvements compared with XO since the per-core ondemand governor determines P-states close to demanded ones by each VM.

Figure 9 plots the 95th percentile latency for memcached with the high-load level; we do not plot for apache since it also shows the similar results with memcached. VIP-selective and VIP-selective-lazy show the similar 95th percentile latency for the runtime compared with XO; note that VIP does not show the difference with XO in terms of throughput (50K pps). Satisfying the SLA with lower P-states obviously improves energy efficiency since the lower P-states consumes lower power [18]. Since the VIP satisfies the SLA with lower P-states set by guest ondemand governor as plotted, the VIP shows the improvements in energy efficiency.

7 RELATED WORK

PM collaborating with guest OS. VirtualPower [25] attempts to preserve each VM's independent PM policy in a processor supporting chip-wide DVFS by the combination of hardware scaling and software scaling adjusting time slice of each VM. While such scheduler-based approach often limits scheduling policies as the number of cores/VMs increases and a VM has multiple vCPUs, VIP is more adaptable and more straightforward since VIP does not rely on a particular scheduling policy. VIP shows further improvements with hypervisor-assisted techniques exploiting scheduling behaviors while not limiting scheduling policy. Stoess et al. [28] propose a framework for energy management that controls energy constraints and supports energy-aware guest OS for application-specific PM. They focus on PM based on energy constraints instead of enabling/supporting heterogeneous PM governors for consolidated VMs.

PM in heterogeneous VM environments. Liu et al. [22] and Hagimont et al. [15] investigate energy-inefficiency problems when heterogeneous VMs are consolidated. Liu et al. conduct an analysis on the interplay between the scheduler and the host ondemand governor [22]. They show potentials of improving energy efficiency when scheduler is enhanced with estimated energy consumptions. Hagimont et al. propose a scheduling scheme to alleviate performance degradation posed by the host ondemand governor [15] through allowing the penalized VM to run longer than its allocated CPU share. VIP also demonstrates energy inefficiency of the host ondemand governor when heterogeneous VMs are consolidated by measuring actual energy consumption.

DVFS collaborating with scheduler. For energy-efficiency, DVFS algorithms collaborating with scheduler and other management mechanism are proposed to improve energy efficiency. Merkel et al. [24] propose to use frequency scaling along with co-scheduling and migration to improve energy efficiency and performance by avoiding resource contention in virtualized environments. Wen et al. [32] propose a power credit-based scheduler by adjusting consumed credits according to frequency to improve fairness of energy consumption along with performance. While VIP does not modify and affect the scheduling policy of the hypervisor, it collaborates with the scheduler and exploits the scheduling information to support per-VM PM efficiently.

Application-level frequency control. The TURBO diaries [31] proposes a library that enables control of frequency from user space by modifying source codes of applications. Thus, it requires code-level knowledge of applications to control frequency effectively. Since VIP enables VMs to control the frequency, such application-level frequency control technique can be deployed in a guest VM with VIP in addition to supporting existing governors offered by guest OSes.

VM power metering. Virtual machine power metering techniques are proposed to measure the power consumption of individual VMs separately [17, 21]. The power metering technique along with VIP supporting per-VM PM can facilitate diverse pricing models based on energy consumption. This allows VMs to deploy a PM policy based on the charge estimated by the power metering techniques. Besides, BITWATTS[11] provides a per-process/application power metering in a single VM. VIP can consider the estimated power consumption of each application when they choose or develop a PM policy efficient for the applications.

Cluster-level PM in VM environments. VMware Distributed Power Management (DPM) [7] considers power consumption when placing and consolidating VMs on a fewer physical machine. vGreen [12], Verma et al. [29], and Laszewski et al.[30] propose power-aware scheduling policies considering performance and power consumption characteristics of VMs. Such cluster-level PMs can be integrated with per-VM PM supported by VIP.

Frequency emulation. Kamga [16] proposes to emulate non-existing frequencies instead of running a core at the next higher frequency for saving energy in virtualized systems. Since the recent processors support many P-states, a guest governor also can effectively improve energy efficiency by exploiting existing P-states offered by the physical processor with VIP while supporting consolidated/overcommitted multi-core environments.

Micro-architectural resource virtualization. vCache [20] virtualizes the last-level cache (LLC) so that each VM utilizes a transparent and isolated virtual LLC in consolidated VM environments. To this end, vCache proposes Guest Physical Address (GPA) based indexing and VM-based cache partitioning mechanisms. Virtual Snooping [19] virtualizes a coherence domain by maintaining a virtual snoop domain per VM exploiting isolation property among VMs. Virtual Snooping filters coherence requests across VM boundaries and propose techniques that address issues caused by imperfect isolation. Single Root I/O Virtualization (SR-IOV) [4, 13] virtualizes the hardware packet queue as a Virtual Function (VF) in Network Interface Card (NIC). The SR-IOV technique provides an isolated and dedicated VF to each VM allowing each VM to utilize the NIC without interference by other VMs and intervention of the hypervisor. As the mentioned studies virtualize micro-architectural resources and techniques current hypervisors overlook, VIP virtualizes P-states and PM governors that allow VMs to deploy its own PM policy and thus improve energy efficiency.

8 CONCLUSION

In this paper, after tackling limitations of current hypervisor's per-core based PM governors, we propose a per-VM power management mechanism, called VIP, that supports **VI**rtual **P**erformance state and enables an independent guest governor for each VM. To this end, VIP first defines virtual P-states per vCPU by exploiting P-state information supported by the physical processor. After enabling guest governors, VIP enforces the virtual P-state of vCPUs maintained in vpstate register when the guest governors attempt to update the vpstate register or vCPUs are scheduled on cores. Consequently, VIP preserves guest PM policies with strong PM isolation in consolidated VM environments. Furthermore, VIP implements hypervisor-assisted mechanisms for more efficient PM after investigating the scheduling behavior of consolidated VMs and limitations of strictly enforcing guest governors with I/O-intensive VMs. Considering the scheduling status of vCPUs, VIP selectively or lazily enforces virtual P-states considering scheduling behaviors that incur frequent vCPU switches, such as boosting technique, in consolidated VM environments. Such per-VM PM enables cloud providers to facilitate more diverse pricing models based on power/energy consumption along with improving energy efficiency in the cloud. Our experimental results show that VIP reduces the overall energy consumption and the execution time of CPU-intensive VMs compared with the Xen's ondemnad governor by up to 27% and 32%, respectively, when heterogeneous VMs share physical cores.

ACKNOWLEDGMENTS

This research was supported in part by a National Research Foundation of Korea (NRF) grant funded by the Korea government (MSIP) (No. NRF-2017R1C1B2008571), an Institute for Information communications Technology Promotion(IITP) funded by the Korea government (MSIP) (No. 2014-0-00065, Resilient Cyber-Physical Systems Research), and an NSF grant (CNS-1557244). Correspondence should be addressed to Daehoon Kim.

REFERENCES

[1] ab - Apache HTTP Server Benchmarking Tool. [Online]. Available: https://httpd.apache.org/docs/2.4/programs/ab.html.
[2] ACPI: Advanced Configuration & Power Interface. [Online]. Available: http://www.acpi.info.
[3] Amazon Elastic Compute Cloud (EC2). [Online]. Available: http://aws.amazon.com/ec2.
[4] PCI Special Interest Group. [Online]. Available: http://www.pcisig.com/home.
[5] SPEC CPU™2006. [Online]. Available: https://www.spec.org/cpu2006.
[6] vSphere. [Online]. Available: http://www.vmware.com/products/vsphere.
[7] VMware inc., VMware distributed power management: Concepts and use. [white paper]. 2010.
[8] M. Alian, A. H. Abulila, L. Jindal, D. Kim, and N. S. Kim. Ncap: Network-driven, packet context-aware power management for client-server architecture. In *In Proceedings of 2017 IEEE International Symposium on High Performance Computer Architecture (HPCA)*, 2017.
[9] D. Ardagna, B. Panicucci, M. Trubian, and L. Zhang. Energy-aware autonomic resource allocation in multitier virtualized environments. *IEEE Transactions on Services Computing*, 5(1):2–19, 2012.
[10] 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 *Proceedings of the 19th Symposium on Operating Systems Principles (SOSP)*, 2003.
[11] M. Colmant, M. Kurpicz, P. Felber, L. Huertas, R. Rouvoy, and A. Sobe. Process-level power estimation in VM-based systems. In *Proceedings of the 10th European Conference on Computer Systems (Eurosys)*, 2015.
[12] G. Dhiman, G. Marchetti, and T. Rosing. vGreen: A system for energy efficient computing in virtualized environments. In *Proceedings of the 2009 International Symposium on Low Power Electronics and Design (ISLPED)*, 2009.
[13] Y. Dong, X. Yang, J. Li, G. Liao, K. Tian, and H. Guan. High performance network virtualization with sr-iov. *Journal of Parallel and Distributed Computing*, 72(11):1471–1480, 2012.
[14] M. Ferdman, A. Adileh, O. Kocberber, S. Volos, M. Alisafaee, D. Jevdjic, C. Kaynak, A. D. Popescu, A. Ailamaki, and B. Falsafi. Clearing the clouds: a study of emerging scale-out workloads on modern hardware. In *ACM SIGPLAN Notices*, volume 47, pages 37–48. ACM, 2012.
[15] D. Hagimont, C. M. Kamga, L. Broto, A. Tchana, and N. De Palma. Dvfs aware cpu credit enforcement in a virtualized system. In *Proceedings of the 14th International Middleware Conference (Middleware)*. 2013.
[16] C. M. Kamga. Cpu frequency emulation based on dvfs. *Proceedings of IEEE 5th International Conference on Utility and Cloud Computing (UCC)*, 2012.
[17] A. Kansal, F. Zhao, J. Liu, N. Kothari, and A. A. Bhattacharya. Virtual machine power metering and provisioning. In *Proceedings of the 1st Symposium on Cloud Computing (SOCC)*, 2010.
[18] H. Kasture, D. B. Bartolini, N. Beckmann, and D. Sanchez. Rubik: Fast analytical power management for latency-critical systems. In *Microarchitecture (MICRO), 2015 48th Annual IEEE/ACM International Symposium on*, pages 598–610. IEEE, 2015.
[19] D. Kim, H. Kim, and J. Huh. Virtual snooping: Filtering snoops in virtualized multi-cores. In *In Proceedings of the 43rd Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*, 2010.
[20] D. Kim, H. Kim, N. S. Kim, and J. Huh. vcache: Architectural support for transparent and isolated virtual llcs in virtualized environments. In *Proceedings of the 48th International Symposium on Microarchitecture (MICRO)*, 2015.
[21] B. Krishnan, H. Amur, A. Gavrilovska, and K. Schwan. VM power metering: feasibility and challenges. *ACM SIGMETRICS Performance Evaluation Review*, 2011.
[22] M. Liu, C. Li, and T. Li. Understanding the impact of vcpu scheduling on dvfs-based power management in virtualized cloud environment. In *Proceedings of the 22nd International Symposium on Modelling, Analysis & Simulation of Computer and Telecommunication Systems (MASCOTS)*, 2014.
[23] A. Merkel, J. Stoess, and F. Bellosa. Resource-conscious scheduling for energy efficiency on multicore processors. In *Proceedings of the 5th European Conference on Computer Systems (Eurosys)*, 2010.
[24] A. Merkel, J. Stoess, and F. Bellosa. Resource-conscious scheduling for energy efficiency on multicore processors. In *Proceedings of the 5th European Conference on Computer Systems (Eurosys)*, 2010.
[25] R. Nathuji and K. Schwan. VirtualPower: coordinated power management in virtualized enterprise systems. In *Proceedings of the 21st Symposium on Operating Systems Principles (SOSP)*, 2007.
[26] V. Pallipadi and A. Starikovskiy. The ondemand governor. In *Proceedings of the Linux Symposium*, 2006.
[27] V. Spiliopoulos, S. Kaxiras, and G. Keramidas. Green governors: A framework for continuously adaptive dvfs. In *Proceedings of International Green Computing Conference and Workshops (IGCC)*, 2011.
[28] J. Stoess, C. Lang, and F. Bellosa. Energy management for hypervisor-based virtual machines. In *Proceedings of the 2007 USENIX conference on Annual Technical Conference (ATC)*, 2007.
[29] A. Verma, P. Ahuja, and A. Neogi. Power-aware dynamic placement of HPC applications. In *Proceedings of the 22nd Annual International Conference on Supercomputing (ICS)*, 2008.
[30] G. Von Laszewski, L. Wang, A. J. Younge, and X. He. Power-aware scheduling of virtual machines in dvfs-enabled clusters. In *Proceedings of IEEE International Conference on Cluster Computing and Workshops (CLUSTER)*, 2009.
[31] J.-T. Wamhoff, S. Diestelhorst, C. Fetzer, P. Marlier, P. Felber, and D. Dice. The TURBO diaries: Application-controlled frequency scaling explained. In *Proceedings of the 2014 USENIX Conference on Annual Technical Conference (ATC)*, 2014.
[32] C. Wen, J. He, J. Zhang, and X. Long. PCFS: power credit based fair scheduler under dvfs for muliticore virtualization platform. In *Proceedings of International Conference on Green Computing and Communications (GreenCom) & International Conference on Cyber, Physical and Social Computing (CPSCom)*, 2010.