Power consumption management under a low-level performance constraint in the Xen hypervisor

Rolando Brondolin
DEIB, Politecnico di Milano
Milano, Italy
rolando.brondolin@polimi.it

Marco Arnaboldi
DEIB, Politecnico di Milano
Milano, Italy
marco.arnaboldi@polimi.it

Marco D. Santambrogio
DEIB, Politecnico di Milano
Milano, Italy
marco.santambrogio@polimi.it

Abstract

Virtualization is the main building block of many architectures and systems from embedded computing to large scale data-centers. Managing efficiently computing resources and their power consumption becomes fundamental to optimize the performance of the workloads running on those systems, however, hardware tools like Intel RAPL can only introduce power caps without considering performance. This paper presents a performance-aware power capping orchestrator for the Xen hypervisor. The tool exploits hybrid power management techniques to minimize power consumption respecting a given SLA, leveraging RAPL and overcoming its limitations. Experimental evaluation shows that the proposed approach guarantees good results for almost all the analyzed benchmarks (e.g. CPU-, memory- and IO-bound).

CCS Concepts **•Software and its engineering →Virtual machines; Power management; •Computer systems organization →Self-organizing autonomic computing;**

Keywords Adaptive Systems; Power Management; Virtualization

1 Introduction

Virtualization is currently the standard technology for the deployment of workloads on a large variety of architectures and computing platforms. From small embedded systems [24] to large scale computing infrastructures and cloud data-centers [21], virtualization provides a clean way to separate tenants co-located on the same physical platform while being transparent w.r.t. the tenant's users. Virtual Machines (VMs) can reach near-native performance thanks to hardware support and the research in the last years focused on how to run VMs efficiently on commodity hardware. This is not only related to pure performance per se, but also on how to manage power consumption as well. This is an important aspect, as power draw accounted directly to the servers represents $\simeq$ 30% of the power consumption of a cloud data-center [7].

To cope with the increasing pressure on power grids and on power supplies in general, Intel introduced the Running Average Power Limit (RAPL) interface since the Sandy Bridge [9] processors generation: this interface enforces a strong and precise limit on the power consumption of a processor, i.e., the component that contributes the most on the *dynamic* power consumption of a server [27]. RAPL uses Dynamic Voltage and Frequency Scaling (DVFS) techniques to guarantee the desired power cap. Although RAPL is a precise and fast solution to reduce power consumption on Intel processors, it is not aware of the impacts that these techniques have on the performances of the hosted applications. To cope with this limitation, software techniques can be combined with RAPL to build hybrid approaches able to manage the trade-off between performance and power at runtime. On the one hand, performance can be maximized under a given power cap, and interesting examples of this approach are *PUPiL* [28] and *XeMPUPiL* [5]. On the other hand, RAPL can be used to minimize power consumption while respecting a given performance constraint. This approach is extremely interesting for workloads that should respect strict Service Level Agreemens (SLAs) to work properly.

In this paper, we propose a hybrid (hardware and software) power capping orchestrator for the Xen hypervisor based on the Observe Decide Act (ODA) control loop that aims at minimizing the power consumption of a physical server given an SLA to respect. The main contributions of this paper are the following:

- we propose an *Observe* phase that collects a low-level and generic performance metric like Instruction Retired (IR) without instrumenting the applications;
- we define a *Decide* phase that deals with virtual resources, assigning them to physical ones to maintain the given SLA and minimizing power consumption;
- we implement an *Act* phase that supports the Xen hypervisor and enforces the results of the previous phases.

The rest of this paper is organized as follows: Section 2 introduces the challenges we are going to target within this work; Section 3 describes the state of the art; Section 4 details the methodology and the proposed contributions; Section 5 presents the experimental results that validate our approach, while Section 6 draws the conclusions and presents the future directions of this work.

2 Problem Definition and goals

The leading technique adopted in data-centers to increase the efficiency of the physical machines is virtualization. Of

EWiLi '19, October 17, 2019, New York, USA.

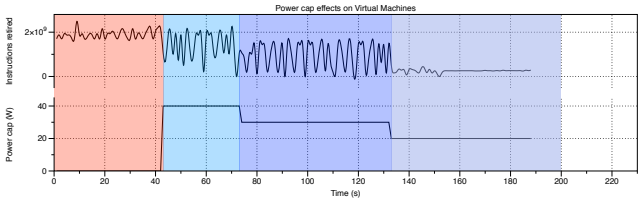

Figure 1. Performance (measured in IR) of a domain in Xen, running an NPB EP benchmark that is affected by increasing power caps.

course, when hardware resources are affected by power constraints, the virtualized ones are affected as well. The graph in Figure 1 shows how different power constraints affect the performance returned by a VM running in a virtualized environment. The application running in the VM is a highly parallel random number generator taken from the NAS Parallel Benchmarks (NPB). It is possible to notice that in a system where no power constrains (region 1) are defined, the performance (measured as the number of IR) is optimal. The challenge rises when a power cap must be enforced on the system as shown in region 2. In this case, it is possible to notice a slight downgrade in performance. This behavior is even more clear when the power cap becomes stricter, like in regions 3 and 4. In these cases, the performance downgrade is really significant. However, it is also true that being able to exploit a power cap technique like the one in the graph (in this case was used RAPL) ensures a precise and strict power control of the system.

The goal of this work consists in the design and development of an orchestrator for workloads running in a virtualized environment able to manage at best performance SLAs and power caps. The proposed orchestrator should be: *timely*, *efficient* and *workload agnostic*. At any change of the power cap for the system, the orchestrator should be able to *timely* enforce it, avoiding any oscillatory behavior. Depending on the different running workloads in the system, the orchestrator should be able to identify *efficiently* which is the best resource set to assign to each workload in order to maximize its performance. Moreover, the system should be *workload agnostic* to be as general as possible without instrumenting code and guest VMs.

3 State of the Art

Power management approaches can be classified into two families: hardware and software approaches. The former ones are built upon the concept of timeliness, enforcing the cap as faster and stricter as possible exploiting hardware control circuits. The latter ones, instead, are built upon the concept of efficiency, searching for the best configuration possible to maximize the performance while reducing the power consumption. Moreover, within these two families, we can find some works that can be defined as hybrid, since they leverage both techniques in such a way that none of the two approaches is prominent.

All the power capping techniques implying the use of socket modules or interfaces to enforce a cap can be classified as hardware approaches. In the work proposed by Deng et al. [10] the authors present MultiScale. This is the first technique that tries to manage DVFS in systems presenting multiple memory channel, devices, and Memory Controllers (MCs). This approach consists into monitoring workload bandwidth requirements across MCs, under Operating System (OS) control. The information retrieved from the monitoring stage is then used by a heuristic in order to select the best frequencies combinations. These combinations try to minimize the overall system power consumption while respecting the user-specified per-application performance constraints. Instead, in the work of Horvath et al.[19], the authors address DVFS in multistage service pipelines, unlike previous works that addressed DVFS on individual servers and on load-balanced server replicas. Finally, for what concerns hardware techniques, the survey published in 2012 by the Intel sandy-bridge development team [25] provides useful information about the new features introduced in the sandy-bridge processors family. The Intel developer manual [15], volume 3B, section 14.9, shows that RAPL can be defined as an interface providing mechanisms to enforce power consumption limit. The usage of those interfaces has huge importance for both client and server platforms.

As for software techniques, Cochran et al. presented Pack & Cap [8], a control technique that aims to maximize performance while respecting a given power budget. This goal is achieved through a system designed to make optimal thread packing control decisions. The work proposed by Hoffman et al. in 2013 [16] showed a detailed analysis over different software heuristics for workloads with deadlines that can be divided into three families: *Race-to-idle*, *Pace-to-idle* and *No-idle* depending on the fact that the workloads obtain the most performing configuration and reach idle as soon as possible, the workloads obtain the most energy-efficient configuration able to respect the deadline or the workloads respect the deadline but never idle respectively. Finally, with model-based software techniques the goal is to reduce power in data-center [13, 14, 19, 22, 26] or in embedded systems [11, 20, 23] by leveraging data-driven models. Models based on previously observed behavior can predict the power and the resources required by a new unobserved application to fine-tune the resources' configuration and save power.

For what concerns hybrid approaches, the work proposed by Zhang and Hoffmann [28] is the most remarkable one. *PUPiL* is an orchestrator for applications in a Linux bare metal OS. It totally embraces the definition of hybrid power consumption management technique. Its approach is composed by a software part (i.e. an ODA control loop) and a hardware one (i.e. Intel RAPL interface), having the goal of maximizing the performance and at the same time strictly

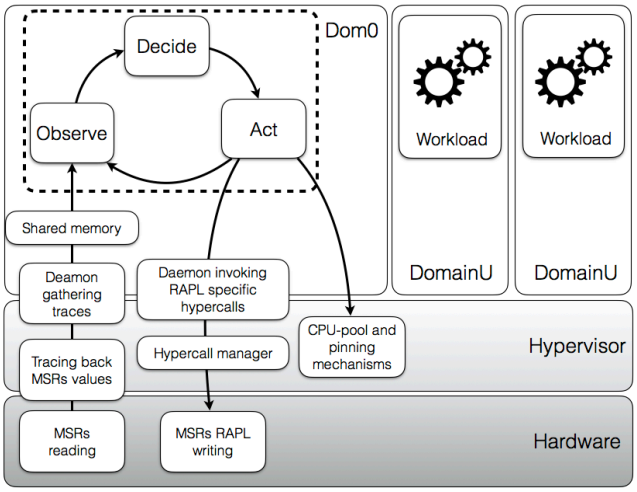

Figure 2. Overview of the orchestrator architecture with observe, decide and act phases. The observe phase collects data at the hardware and hypervisor level, while the act phase leverages RAPL to enforce power caps and CPU-pool and pinning mechanism for resource management.

respecting a given power cap. Even though the approach proposed by *PUPiL* is effective, we identified two non-negligible limitations of the proposed solution: first, the applications running on the system need to be instrumented with the *Heartbeat framework* [17, 18], in order to provide a uniform metric of throughput to the decision phase; second, the tool is meant to work with applications running bare-metal on Linux. To overcome this limitation, Arnaboldi et al. presented *XeMPUPiL*[5], which leverages IR as a common metric and supports virtualization in the *Xen hypervisor* [6]. This paper starts from *XeMPUPiL* and devices an orchestrator able to guarantee the performance expressed as number of IR while minimizing the server power consumption.

4 Methodology

In this paper we present a hybrid power-aware orchestrator that exploits software and hardware techniques to achieve power control over the system. This work extends ant takes inspiration from *XeMPUPiL*[5], enabling to express a low-level performance constraint and reducing power consumption whenever possible. We target a virtualized environment such as Xen, addressing all the problems and the challenges related to the isolation between the running jobs inside the guest OS and the virtualized hardware.

4.1 System architecture

The overall architecture of the proposed orchestrator is shown in Figure 2. At the bottom, there is the hardware layer, where the physical hardware resources lie. On top of this level, there is the hypervisor, which is in charge of the virtualization of the underlying resources for the domains level. In this last level, the domains (i.e. the VM) are instantiated and can exploit the virtualized resources provided by the hypervisor. In this layer, the domains containing the workloads will be instantiated and executed.

The orchestrator is built upon an ODA control loop. It *observes* the running workloads and how they are performing in terms of power consumption and performance, thanks to hardware performance metrics. It then explores the space of all the feasible configurations and *decides* the one providing the required SLA (expressed as IRs) while saving as much power as possible. Finally, it enforces the selected configuration through the *act* stage. To maintain this approach as portable as possible, we decided to implement the orchestrator logic at the highest level possible. To this aim, we inflated the control logic inside *dom0*, given that this VM is the first one instantiated in *Xen* every time the hypervisor is initialized and provides privileged access to the hypervisor.

4.2 Observe

The main challenge of the *Observe* phase consists of avoiding any code or domain instrumentation. In this way we can obtain an approach as general and portable as possible, avoiding additional effort of the application developers. We decided to use hardware event counters as low-level metrics of performance, exploiting the Intel Performance Monitoring Unit (PMU) to monitor the amount of IR accounted to each domain in a certain time window. We chose as main metric the IRs because they measure how many instructions were completely executed (i.e., that successfully reached the end of the pipeline) between two samples of the counter, thus representing a reasonable indicator of performance [1].

There are three challenges that we should address in this context: (i) provide precise attribution of hardware events to virtual tenants; (ii) be agnostic to the mapping between virtual and physical resources, hosted VMs and scheduling policies; (iii) add negligible overhead. To address these challenges we leveraged and modified *XeMPower* [12], a tool already part of Xen developed by Politecnico di Milano in collaboration with the SwarmLab at UC Berkeley. We instrumented the Xen scheduler in order to read IRs for each domain from the hardware registers and to empty them at each context switch. Then we exploited the daemon provided by *XeMPower* to gather the information coming from the scheduler, aggregating data per domain and storing them in a shared-memory region read by the observe phase. Finally, the data is sent to the *Decide* phase to drive the current decision policy.

4.3 Decide

During this step we work with the concept of resource, defining how to assign different resources to a workload. A resource is a computational asset which can modify the performance of the observed applications. In our case, we will manage as a resource the number of virtual CPUs (vCPUs) pinned

over the physical CPUs (pCPUs). We made two assumptions during the development of this work: the performance function is a concave function and the power consumption is a convex function. The first assumption ensures to find a global maximum, hence a unique point of termination. The second one ensures to find a global minimum.

To find the configuration that allows to guarantee a given level of IRs for a given domain while saving power, the following procedure is adopted: (i) a domain is executed without imposing a power cap, (ii) the decide phase finds the resource configuration for the workloads providing the best performance, (iii) the decide phase reduces the power consumption until the SLA is respected.

When the decide phase starts, we set all the possible configurations as not explored and we set up a binary search. Initially, we explore always three configurations: (i) the one with minimum resources, (ii) the one with maximum resources and (iii) the one with an average amount of resources. For each of them, we observe the behavior of the workload, then we define the best lower and upper bound for the next iteration of the binary search. After testing these configurations, the search continues until the upper and lower bound are the same (convergence) or the performance of the new bounds are worse than the ones of the previous iteration. In this second case, the best between the previous ones is defined as the convergence configuration.

For what concerns the allocation of resources to each domain, we chose to work at a core-level granularity: on the one hand, each domain owns a set of vCPU, while, on the other hand, we have a set of pCPU present on the machine. Each vCPU is mapped on a pCPU for a certain amount of time, while it may happen that even multiple vCPU can be mapped on the same pCPU. We wanted our allocation policy to be as fair as possible, covering the whole set of pCPU if possible; given a workload with M virtual resources and an assignment of N physical resources, to each pCPU we assign:

$$vCPUs(i) = \left\lceil \frac{M - \sum_{j=0}^{i-1} vCPUs(j)}{N - i} \right\rceil \quad (1)$$

where i is an integer between 0 and $N - 1$, i.e., it spans over the set of pCPU. This formula represents the following behaviour: if the system has 3 pCPU and a workload has 8 vCPU, (1) leads to a pinning of the vCPUs over the pCPUs of 3-3-2: three vCPU pinned on the first, three on the second and two over the third pCPU.

4.4 Act

The act phase is the step of the ODA loop where the hybrid approach is finally enforced, and essentially consists in: (i) setting the desired power cap; (ii) actuating the selected resource configuration. On the one hand, we decided to implement the same hardware technique proposed by *PUPiL* and *XeMPUPiL* to set the power cap, i.e., exploiting the Intel RAPL interface. This provides a fast and strict response to power oscillations, cutting the frequency and the voltage of the whole CPU socket and ignoring the performance of the applications running on the system. On the other hand, we support the knobs made available by the hypervisor to assign resources to each domain. This second step allows fine-tuning of the resources to improve domains performance, but it is, of course slower than the hardware actuation in responding to power variations. This is the reason why we use both the approaches to provide a fast response and a precise power capping.

4.4.1 Hardware Power Cap

To support hardware power cap, we leveraged the actuation step of *XeMPUPiL* . The power cap is enforced through RAPL by writing data into the `MSR_RAPL_POWER_UNIT` and `MSR_PKG_RAPL_POWER_LIMIT` Model Specific Registers (MSRs) of the processor. The first register contains data about time, energy and power units. The second register is used to write a limit on the power consumption of the socket, where the values are scaled based on the contents of the first register.

XeMPUPiL and *XeMPower* provide two hypercalls that allow to read and write MSRs from *dom0*, operation that was previously not supported by *Xen*. The two hypercalls are `"xempower_rdmsr"` and `"xempower_wrmsr"`, where the first one allows to read, while the second one allows to write a specific MSR from *dom0*. We then set two *Xen* built-in functions as callbacks for the hypercalls: `wrmsr_safe` and `rdmsr_safe`. Finally, we leveraged the Command Line Interface (CLI) tools `xempower_RaplSetPower` to set and `xempower_RaplPowerMonitor` to read the power consumption of the socket.

In a single socket architecture it is straightforward to set the correct value in the right MSR, since it is ensured that, independently from which core the hypercall is executed, the core will belong to the socket. Hence, the MSR hypercall will control the entire socket. However, in a multi-socket environment, it is impossible to predict on which core the hypercall management routine will be executed. For this reason, we modified the `"xempower_wrmsr"` hypercall routine to execute the hypercall on all the online CPUs. We then defined the function that writes the MSR as a *tasklet*. In Xen, tasklets are dynamically-allocatable tasks run in either vCPU context (specifically, the idle VCPU's context) or in softirq context, on at most one CPU at a time. We then exploited a Symmetric MultiProcessing (SMP) call to launch a tasklet on each pCPU. SMP involves a multiprocessor computer hardware and software architecture where two or more identical processors are connected to a single, shared main memory, have full access to all I/O devices, and are controlled by a single operating system instance that treats all processors equally, reserving none for special purposes. This allowed

us to continue the hypercall on a given CPU with hypercall privileges, ensuring the correct update of the MSR.

4.4.2 Software resource management

The main tool we leverage for software resource management is *cpupool*, which is a *Xen* tool part of the *xl* CLI. This tool allows to create clusters of CPUs and to put them into pools. When we declare a pool, a *Xen* domain can use it. In this case, the hypervisor instantiates a new scheduler to manage the pool. At this point, the domain vCPUs can be scheduled onto the pCPUs inside the pool. When a new allocation is defined by the *decide* phase, we leverage *cpupool* to add or remove pCPUs accordingly and we pin the vCPUs of the domain to the pCPUs to increase workload stability.

5 Experimental Evaluation

In this section we will present the results obtained with the proposed orchestrator. We studied its behavior by looking at the resources assigned to the different workloads and the power consumption obtained for the system. We defined different level of SLAs, which correspond to fractions of the best performance defined as the number of IRs obtained after the convergence of the maximization process.

5.1 Experimental setup

For the benchmarking activity, we decided to exploit four different benchmarks, each one representing a possible family of computational workloads having some bounds directly related to the resources of the system. In particular, we decided to investigate the following families: (i) CPU-bound, (ii) memory-bound, (iii) IO-bound and (iv) CPU-mem-bound workloads. Two of them are part of the set of benchmarks provided by the National Aeronautics and Space Administration (NASA), the NPB version 3.3 [3]: Embarassingly parallel (EP) and Block Tridiagonal solver (BT). The former provides estimates for the upper achievable limits for floating point performance. This can be defined as a CPU-bound workload. The latter, instead, is a pseudo application. In detail, it provides solution of different, independent system of block tridiagonal, non-diagonally dominant equations. This application is both parallel and memory bounded. A third benchmark used to represent the IO-bound family is IOzone [2], a tool for benchmarking file-systems. A variety of file operation is measured and generated by this benchmark. Finally, the fourth benchmark is Cachebench [4], a test designed to stress memory and cache bandwidth performance.

The experiments were carried out on a dual-socket Intel(R) Xeon(R) CPU E5-2680 v2 @ 2.80 GHz with 10 physical thread per socket and hyper-threading enabled. We leveraged Xen-4.6 as the base hypervisor. Each benchmark runs isolated inside a VM and we carried out the evaluation using one VM per experiment. The VMs run a 32-bit version of Debian, and for this reason, the maximum amount of vCPUs that the VM is able to manage is up to 8.

5.2 Experimental results

To evaluate our approach, we defined three SLAs: 90%, 80%, and 70%, since we noticed that at these steps it is possible to see a measurable decrease in power consumption. The meaning of these values corresponds to the percentage of the maximum performance that must be at least returned. For instance, given a maximum performance of 1000 IRs obtained thanks to the maximization phase, defining for it a SLA of 80% means trying to minimize the power consumption until the performance downgrades under an absolute value of 800 IRs. Recalling Section 4, the tests on the four benchmarks were conducted as follows:

1. The power cap is imposed to default value. In this way is like imposing no power cap, since the maximum consumption for the sockets of the system is around 160W;
2. The maximization of the performance in a NO-CAP configuration is ran in order to find the resource configuration for the workloads providing the best performance;
3. The minimization of the power consumption is executed in order to respect the given SLA.

In Table 1 the results obtained for the EP benchmark are shown. It is possible to notice that in all the SLA cases, the maximization phase tends to converge to a configuration that assigns all the available resources to the VM. In these cases it is possible to notice that trying to respect a small SLA will result in a greater standard deviation for what concerns the converged performance percentage, meaning that the orchestrator tends to disrespect the SLA defined in case it is too small. A similar behavior can be noticed from Table 2 with BT. In this test case, the application is at the same time computational intensive and memory intensive and this is why fewer cores are assigned to it. This leads to trying to assign to it less pCPUs, due to the memory intensive nature of the application.

Instead, in Table 3 and 4 two benchmarks respectively memory and IO intensive are presented (e.g. Cachebench and IOzone). In this cases, our methodology tends to assign fewer cores to the VMs. And it also possible to notice that the power consumption is decreased significantly. On the other hand, these results present a huge standard deviation. This is due to the IR metric adopted in order to evaluate the performance, since is really a low-level measurement, very sensitive to each phase of the running application, even the smallest one. One last consideration regards the time needed in order to converge to the solution. This value is pretty stable for all the benchmark classes and it is required only once for each iteration. Furthermore, this time can be further decreased since at the moment a time window of 2 seconds is left to the workloads to stabilize when the performance maximization phase is completed.

Table 1. Minimization results obtained for the EP benchmark. Maximum power consumption 160W.

SLA			Cores assigned		Power consumption (W)		Maximization time (s)		Minimization time (s)	
Objective	*Obtained*	*Standard Deviation*	*Mean*	*Standard Deviation*	*Mean*	*Standard Deviation*	*Mean*	*Standard Deviation*	*Mean*	*Standard Deviation*
90	0,97	0,07	7,93	0,25	140,17	40,38	3,73	0,32	5,51	0,78
80	0,94	0,12	7,97	0,18	139,00	42,76	3,57	0,36	5,33	0,58
70	0,82	0,23	7,87	0,43	112,83	54,94	3,61	0,30	5,28	0,67

Table 2. Minimization results obtained for the BT benchmark. Maximum power consumption 160W.

SLA			Cores assigned		Power consumption (W)		Maximization time (s)		Minimization time (s)	
Objective	*Obtained*	*Standard Deviation*	*Mean*	*Standard Deviation*	*Mean*	*Standard Deviation*	*Mean*	*Standard Deviation*	*Mean*	*Standard Deviation*
90	0,99	0,04	7,10	0,92	159,17	4,56	4,40	0,43	5,86	0,86
80	0,96	0,12	7,00	1,23	154,67	18,71	4,39	0,62	5,85	1,09
70	0,91	0,17	7,10	0,92	147,17	32,13	4,51	0,51	5,95	0,94

Table 3. Minimization results obtained for the CacheBench benchmark. Maximum power consumption 160W.

SLA			Cores assigned		Power consumption (W)		Maximization time (s)		Minimization time (s)	
Objective	*Obtained*	*Standard Deviation*	*Mean*	*Standard Deviation*	*Mean*	*Standard Deviation*	*Mean*	*Standard Deviation*	*Mean*	*Standard Deviation*
90	1,00	0,05	5,60	2,40	147,17	35,40	4,51	0,50	6,04	0,79
80	0,99	0,12	5,43	2,28	135,50	46,37	4,62	0,50	6,27	0,93
70	1,00	0,05	5,10	2,09	147,67	35,10	4,68	0,44	6,48	0,99

Table 4. Minimization results obtained for the IOzone benchmark. Maximum power consumption 160W.

SLA			Cores assigned		Power consumption (W)		Maximization time (s)		Minimization time (s)	
Objective	*Obtained*	*Standard Deviation*	*Mean*	*Standard Deviation*	*Mean*	*Standard Deviation*	*Mean*	*Standard Deviation*	*Mean*	*Standard Deviation*
90	0,93	0,11	6,70	1,62	133,33	42,23	4,82	0,55	6,38	1,48
80	0,84	0,18	6,13	2,13	126,33	45,45	4,83	0,43	6,37	0,98
70	0,83	0,26	6,67	1,65	126,00	48,63	4,81	0,49	6,53	1,41

6 Conclusion and Future works

In this paper we presented a performance-aware power capping orchestrator for the Xen hypervisor. We extended the current implementation of *XeMPUPiL*[5] to guarantee a given SLA expressed as IRs while reducing power consumption in a virtualized environment based on the Xen hypervisor. The methodology proposed in this work leverages three main concepts: (i) *efficiency*, (ii) *timeliness*, and (iii) *lack of workload instrumentation*. Within the experimental campaign we showed that the proposed work is able to reduce power consumptions in almost all the benchmark classes tested (e.g., CPU-, memory- and IO-bound ones). Future work of this paper will address the development of an improved decision algorithm able to reduce the duration of the *decide* phase. In particular, it would be interesting to study how different decision policies may influence the convergence time of the software approach to the configuration providing the highest performance.

References

[1] Clockticks per instructions retired (cpi). https://software.intel.com/en-us/node/544403. Accessed: 2016-06-01.

[2] Iozone. http://www.iozone.org. Accessed: 2017-03-15.

[3] Nas parallel benchmarks. http://www.nas.nasa.gov/publications/npb.html#url. Accessed: 2017-03-15.

[4] Openbenchmarking.org. https://openbenchmarking.org/test/pts/cachebench. Accessed: 2017-03-15.

[5] M. Arnaboldi, M. Ferroni, and M. D. Santambrogio. Towards a performance-aware power capping orchestrator for the xen hypervisor. *ACM SIGBED Review*, 15(1):8–14, 2018.

[6] P. R. Barham, B. Dragovic, K. A. Fraser, S. M. Hand, T. L. Harris, A. C. Ho, E. Kotsovinos, A. V. Madhavapeddy, R. Neugebauer, I. A. Pratt, and A. K. Warfield. Xen 2002. Technical report, 2002.

[7] L. A. Barroso, J. Clidaras, and U. Hölzle. The datacenter as a computer: An introduction to the design of warehouse-scale machines. *Synthesis lectures on computer architecture*, 8(3):1–154, 2013.

[8] R. Cochran, C. Hankendi, A. K. Coskun, and S. Reda. Pack & cap: adaptive dvfs and thread packing under power caps. In *Proceedings of the 44th annual IEEE/ACM international symposium on microarchitecture*, pages 175–185. ACM, 2011.

[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 (ISPLED)*, 2010.

[10] Q. Deng, D. Meisner, A. Bhattacharjee, T. F. Wenisch, and R. Bianchini. Multiscale: memory system dvfs with multiple memory controllers. In *Proceedings of the 2012 ACM/IEEE international symposium on Low power electronics and design*, pages 297–302. ACM, 2012.

[11] M. Ferroni, A. Cazzola, D. Matteo, A. A. Nacci, D. Sciuto, and M. D. Santambrogio. Mpower: Gain back your android battery life! In *Proceedings of the 2013 ACM conference on Pervasive and ubiquitous computing adjunct publication*, pages 171–174. ACM, 2013.

[12] M. Ferroni, J. A. Colmenares, S. Hofmeyr, J. D. Kubiatowicz, and M. D. Santambrogio. Enabling power-awareness for the xen hypervisor. *ACM SIGBED Review*, 15(1):36–42, 2018.

[13] M. Ferroni, A. Corna, A. Damiani, R. Brondolin, J. A. Colmenares, S. Hofmeyr, J. D. Kubiatowicz, and M. D. Santambrogio. Power consumption models for multi-tenant server infrastructures. *ACM Transactions on Architecture and Code Optimization (TACO)*, 14(4):38, 2017.

[14] M. Ferroni, A. Corna, A. Damiani, R. Brondolin, J. D. Kubiatowicz, D. Sciuto, and M. D. Santambrogio. Marc: A resource consumption modeling service for self-aware autonomous agents. *ACM Transactions on Autonomous and Adaptive Systems (TAAS)*, 12(4):21, 2017.

[15] P. Guide. *Intel® 64 and IA-32 Architectures Software Developerfis Manual*, 2011.

[16] H. Hoffmann. Racing and pacing to idle: an evaluation of heuristics for energy-aware resource allocation. In *Proceedings of the Workshop on Power-Aware Computing and Systems*, page 13. ACM, 2013.

[17] H. Hoffmann, J. Eastep, M. D. Santambrogio, J. E. Miller, and A. Agarwal. Application heartbeats: A generic interface for expressing performance goals and progress in self-tuning systems. In *4th Workshop on Statistical and Machine learning approaches to ARchitecture and compilaTion (SMART)*, 2010.

[18] H. Hoffmann, J. Eastep, M. D. Santambrogio, J. E. Miller, and A. Agarwal. Application heartbeats for software performance and health. Technical report, August 2009.

[19] T. Horvath, T. Abdelzaher, K. Skadron, and X. Liu. Dynamic voltage scaling in multitier web servers with end-to-end delay control. *IEEE Transactions on Computers*, 56(4), 2007.

[20] M. Kim, M.-O. Stehr, C. Talcott, N. Dutt, and N. Venkatasubramanian. xtune: A formal methodology for cross-layer tuning of mobile embedded systems. *ACM Transactions on Embedded Computing Systems (TECS)*, 11(4):73, 2012.

[21] R. Kumar and S. Charu. Comparison between cloud computing, grid computing, cluster computing and virtualization. *International Journal of Modern Computer Science and Applications*, 3(1):42–47, 2015.

[22] D. Meisner, C. M. Sadler, L. A. Barroso, W.-D. Weber, and T. F. Wenisch. Power management of online data-intensive services. In *Computer Architecture (ISCA), 2011 38th Annual International Symposium on*, pages 319–330. IEEE, 2011.

[23] S. Mohapatra, R. Cornea, H. Oh, K. Lee, M. Kim, N. Dutt, R. Gupta, A. Nicolau, S. Shukla, and N. Venkatasubramanian. A cross-layer approach for power-performance optimization in distributed mobile systems. In *Parallel and Distributed Processing Symposium, 2005. Proceedings. 19th IEEE International*, pages 8–pp. IEEE, 2005.

[24] D. Rossier. Embeddedxen: A revisited architecture of the xen hypervisor to support arm-based embedded virtualization. *White paper, Switzerland*, 2012.

[25] E. Rotem, A. Naveh, A. Ananthakrishnan, E. Weissmann, and D. Rajwan. Power-management architecture of the intel microarchitecture code-named sandy bridge. *Ieee micro*, 32(2):20–27, 2012.

[26] K. Shen, A. Shriraman, S. Dwarkadas, X. Zhang, and Z. Chen. Power containers: An os facility for finegrained power and energy management on multicore servers. In *IEEE 3rd International Conference on Cyber-Physical Systems, Networks, and Applications*. IEEE, 2015.

[27] C. Xu, Z. Zhao, H. Wang, and J. Liu. On the interplay between network traffic and energy consumption in virtualized environment: An empirical study. In *2014 IEEE 7th International Conference on Cloud Computing*, pages 392–399, June 2014.

[28] H. Zhang and H. Hoffmann. Maximizing performance under a power cap: A comparison of hardware, software, and hybrid techniques. pages 545–559, 2016.