Towards a performance-aware power capping orchestrator for the Xen hypervisor*

Marco Arnaboldi
Politecnico di Milano
marco1.arnaboldi@mail.polimi.it

Matteo Ferroni
Politecnico di Milano
matteo.ferroni@polimi.it

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

ABSTRACT

In the last few years, multi-core processors entered into the domain of embedded systems: this, together with virtualization techniques, allows multiple applications to easily run on the same System-on-Chip (SoC). As power consumption remains one of the most impacting costs on any digital system, several approaches have been explored in literature to cope with power caps, trying to maximize the performance of the hosted applications. In this paper, we present some preliminary results and opportunities towards a performance-aware power capping orchestrator for the Xen hypervisor. The proposed solution, called XeMPUPiL, uses the Intel Running Average Power Limit (RAPL) hardware interface to set a strict limit on the processor's power consumption, while a software-level Observe-Decide-Act (ODA) loop performs an exploration of the available resource allocations to find the most power efficient one for the running workload. We show how *XeMPUPiL* is able to achieve higher performance under different power caps for almost all the different classes of benchmarks analyzed (e.g., CPU-, memory- and IO-bound).

Categories and Subject Descriptors

H.4 [**Cloud Computing, Adaptive Systems, Power Management, Virtualization**]

1. INTRODUCTION

Computing systems changed considerably in the last few decades [20]: multi-core processors entered into the domain of embedded systems, allowing multiple embedded applications to run on the same SoC in a wide range of application fields, like automotive, Internet TV, mobile and other embedded use cases like low-power microservers for lightweight scale-out workloads [2, 15]. On the one hand, this improves overall resources utilization, while, on the other hand, some applications can obtain performance improvements via greater concurrency and parallelism.

In this context, *virtualization* enables to run multiple applications on the same physical resources, still ensuring strong isolation to each of them [23, 7]. Unfortunately, this can only lead to a better utilization of the hardware platforms if the hypervisor is able to perform a good resource allocation of the tenants [16, 25]. This task is made difficult by both hardware and software *heterogeneity*: a standard behavior can not be always defined *"a priori"*, as different systems may not be equipped with the same amount of memory and processors, as well as different tenants may be characterized by different workload profiles (e.g., memory-bound, I/O-bound and/or CPU-bound).

Moreover, this scenario gets even worse when considering power consumption, a major concern for almost every digital system. As embedded devices may be power-constrained or even battery-powered, tools and interfaces need to be introduced to control and limit power consumption, i.e., to set a *power cap*. To face this first requirement, Intel introduced the RAPL interface since its second generation of Sandy Bridge processors [10]: 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 common workstation[26].

RAPL uses Dynamic Voltage and Frequency Scaling (DVFS) techniques to guarantee the desired power cap but is not aware of the impacts that these have on the performances of the hosted applications. Of course, these performances need to be maximized even when a power cap is enforced: we want to find the most *power efficient* hardware configuration under a certain power cap, thus maximizing the performance-per-watt ratio. In order to accomplish our goal, a uniform metric of performance has to be defined, as well as a smart orchestration policy to guarantee the stability of the system as soon as its runtime conditions change.

In this paper, we propose *XeMPUPiL* , a hybrid hardware and software power capping orchestrator for the Xen hypervisor, based on the PUPiL ODA control loop [27], that aims at maximizing the performance of a workload under a power cap. The main contributions of this work are the following:

1. we propose an *Observe* phase that takes into account a generic performance metric for all the hosted tenants, avoiding any instrumentation of the workloads;
2. we improved the decision phase of PUPiL, to deal

*EWiLi'16, October 6th, 2016, Pittsburgh, USA.

with the resources available in a multi-tenant virtualized environment;

3. we implemented a new *Actuation* phase, to support all the *knobs* that Xen provides to control the resources assigned to each tenant.

The rest of the paper is organized as follows: Section 2 discusses some related work, while Section 3 presents the proposed approach and some implementation details; preliminary results are detailed in Section 4, discussing the limitations of this work in Section 5, finally drawing some conclusions in Section 6.

2. RELATED WORK

Several works in the literature propose different approaches to both performance maximization under a power cap and power consumption minimization under performance constraints. For instance, some of them exploit DVFS techniques and try to pack together similar threads [9], while others try to minimize the times the cores go into idle states, in order to save the power spent in going from an idle state back to an active one [18]. Most of these works aims at reducing costs in data centers [14, 21, 24] or to increase battery life in power-constrained devices [19, 22, 11], while our main focus is performance maximization under a strict power cap.

A remarkable work with our same goal is PUPiL, a framework that aims to minimize and to maximize respectively the concept of timeliness and efficiency: *timeliness* is intended as the ability of the system in enforcing a new cap, while *efficiency* is meant as the performance delivered by the applications under a fixed power cap [27]. In order to achieve these goals, PUPiL exploits both hardware (i.e., the Intel RAPL interface [10]) and software (i.e., resource partitioning and allocation) techniques inside a canonical ODA control loop, one of the main building blocks of *self-aware computing*.

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* [13, 12], 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. Both these conditions might not be met in the context of a multi-tenant virtualized environment, in which a virtualization layer allows the execution of multiple workloads and ensures isolation to each of them. This is the case of the *Xen hypervisor* [8], a bare-metal type-1 hypervisor widely adopted in real production environments [6], that runs directly as an abstraction layer between the hardware and the hosted virtual machines, called *domains* in the Xen terminology. It is based on a microkernel design, providing services that allow multiple operating systems to concurrently run on the same hardware. A privileged domain, called *Dom0*, is in charge of managing the *DomU* unprivileged domains. In this context, the high isolation of each tenant, seen as a *black box*, makes

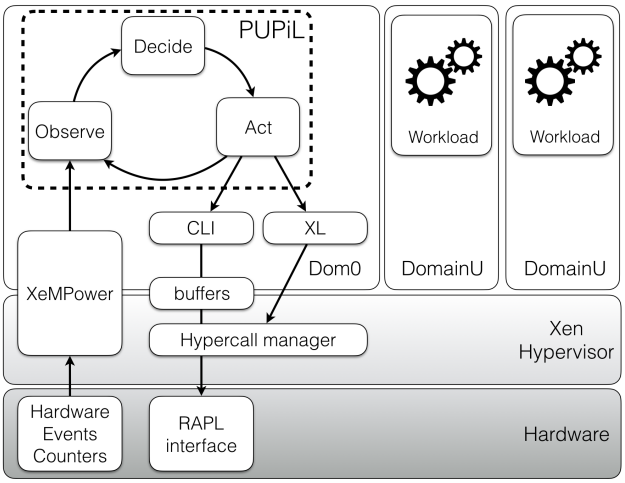

Figure 1: Overview of the proposed approach

any instrumentation of the code of the hosted applications not feasible in a real production environment.

In this paper, we want to extend the current implementation of PUPiL[1] to make it work in a virtualized environment based on the Xen hypervisor, without requiring any instrumentation of the guest workloads, as discussed in the next sections.

3. SYSTEM DESIGN AND IMPLEMENTATION

XeMPUPiL is a hybrid hardware and software power capping orchestrator for the Xen hypervisor. It is *hybrid* as it makes use of the RAPL *hardware* interface to set a strict limit on the processor's power consumption, while a *software*-level ODA loop structure performs an exploration of the available resource allocations, to find the most power efficient one for the running workload. Of course, the innovation does not lie in the exploitation of the well-known ODA loop structure, but in the adoption of an hybrid power capping approach in a virtualized environment.

An overview of the system is presented in Figure 1. Each different phase of the ODA loop needs to interact with different tools throughout all the layers of the stack: some tools are available in Dom0, while other APIs are provided by specific hypercalls to the Xen hypervisor, that allows *XeMPUPiL* to set the domains configurations and guarantees a controlled access to the underlying hardware.

In more detail, a brief description to the high-level flow is here given:

- *XeMPUPiL observes* the power consumption of the system and a set of hardware events of interest for each running domain;

[1]All source code, scripts, inputs, and patches are available at: https://github.com/PUPiL2015/PUPIL.git

- the traced events are then used as metrics of performance, in order to *decide* which hardware configuration is the most power efficient for the current workload;
- finally, the *actuation* phase sets the system to the best configuration found, to maximize the performance under the desired power cap enforced through the RAPL interface.

In this section, we present the design and the implementation of the three ODA loop phases, describing the limitations faced while working in a virtualized environment.

3.1 Observe

This first phase is in charge of monitoring the system and the hosted domains, gathering all the information needed by the subsequent *decide* phase.

As stated in the previous sections, we need to choose a uniform metric of performance without any instrumentation of the guest workloads: each domain remains a black box to the hypervisor, as well as by the other domains (e.g., Dom0 itself). We decided to use hardware event counters as low level metrics of performance, exploiting the Intel Performance Monitoring Unit (PMU) to monitor the number of Instruction Retired (IR) accounted to each domain in a certain time window. Among all the available hardware events that can be monitored, we chose to count the IR events on purpose, because these give an insight on how many micro-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, as the same manufacturer suggests in [1].

In order to monitor these hardware events, we chose to use the *XeMPower* tool[2], a lightweight monitoring solution for the Xen hypervisor designed to: (i) provide precise attribution of hardware events to virtual tenants, (ii) be agnostic to the mapping between virtual and physical resources, hosted applications and scheduling policies, and (iii) add negligible overhead. It uses hypervisor-level instrumentation to monitor every context switch between domains and it does not require any instrumentation of the code of the workload, a strong requirement of the approach proposed in this paper. We patched the *XeMPower* tool to provide *XeMPUPiL* the amount of IR counted for each running domain over the last second : more details on how we use this rate are provided in the following section.

3.2 Decide

The *decision* phase is similar to the one implemented in PUPiL. The major changes are in how we evaluate the metrics gathered in the previous phase and in how we assign the physical resources to each virtual domain.

[2]Source code has been made available at: https://bitbucket.org/necst/xempupil

The evaluation criterion is based on the average IR rate measured over a certain time window: this allows the workload to adapt to the actual configuration in that time window before a new decision is taken. The comparison of the two IR rates highlights which one makes the workload perform better, thus discarding the worse one.

Once the configuration has been chosen, the second part of the decision phase begins: it 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 virtual CPUs (vCPUs), while, on the other hand, we have a set of physical CPUs (pCPU) present on the machine. Each vCPU can be mapped on a pCPU for a certain amount of time, while it may happen that multiple vCPUs can be mapped on the same pCPU.

We wanted our allocation policy to be as fair as possible, covering the whole set of pCPUs if possible; given a workload with M virtual resources and an assignment of N physical resources, to each $pCPU_i$ we assign:

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

where i is an integer between 0 and $N-1$, i.e., it spans over the set of pCPUs.

3.3 Act

The *act* phase essentially consists in: 1) setting the desired power cap and 2) actuating the selected resource configuration.

On the one hand, we decided to implement the same hardware technique proposed by PUPiL to set the power cap, i.e., exploiting the Intel RAPL interface. This provides a fast and strict response to power oscillations, harshly cutting the frequency and the voltage of the whole CPU socket, ignoring the performance of the applications actually running on the system.

On the other hand, we had to support the *knobs* made available by the hypervisor to assign resources to each domain. This second step allows a 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, still trying to find the best resource allocation to maximize the performance of each domain under the power cap.

3.3.1 Hardware power cap

A bare metal operating system can easily access the RAPL interface to set a power cap on the system by writing data into the right Model Specific Register (MSR) of the processor. The two registers of interest to our purposes are:

- `MSR_RAPL_POWER_UNIT`
- `MSR_PKG_RAPL_POWER_LIMIT`

The former contains processor-specific time, energy and power units, used to scale each value read or written on the RAPL MSR, in order to obtain a valid power or energy measure; the latter can be written to set a limit on the power consumption of the whole CPU socket.

In a virtualized environment, these registers are not directly accessible by the virtual domains, even from the privileged tenant Dom0. However, this limitation can be overcome by invoking custom hypercalls that can directly access the underlying hardware. To the best of our knowledge, the Xen hypervisor does not natively support specific hypercalls to interact with the RAPL interface: as a consequence, we implemented our custom hypercalls to this purpose. In order to be generic enough, we implemented two hypercalls: `"xempower_rdmsr"` and `"xempower_wrmsr"`. The first one allows to read, while the second one allows to write a specified MSR from Dom0.

Each hypercall needs to be declared inside the kernel of the hypervisor, that runs bare metal on the hardware. The kernel keeps track of the list of hypercalls available and the input parameters they accept. For each of them, a callback function has to be declared and implemented to be accessible by the kernel at runtime: our implementation makes use of two Xen build-in functions to safely read and write MSR registers, i.e., `wrms_safe` and `rdmsr_safe`; these raise exceptions if something goes wrong in accessing the registers, avoiding errors and faults to undermine the kernel stability.

We then implemented our own Command Line Interface (CLI) tools to access these hypercalls from Dom0:

- textttxempower_RaplSetPower to set;
- `xempower_RaplPowerMonitor` to read the power consumption of the socket.

Arguments (e.g., the desired value of power cap and the power consumption measured) are passed through the whole stack using a set of buffers that allow a fast and safe communication between different hierarchical protection domains [17] (i.e. ring0 for Xen and ring3 for Dom0). The CLI tools are in charge of performing some checks on the input parameters, as well as of instantiating and invoking the Xen command interface to launch the hypercalls.

3.3.2 Software resource management

The current implementation of *XeMPUPiL* exploits two tools provided by the Xen hypervisor to tune the performance and assign resources to domains.

The first one is the *cpupool* tool: this is part of the Xen *xl* CLI and allows to cluster the physical CPUs in different pools. Once a pool is declared, it is possible to create a domain that uses that pool: a new scheduler is instantiated in order to manage the pool. It will then schedule the domain's vCPUs only on the pCPUs that are part of that cluster. Our approach exploits this tool to assign more pCPUs to a domain at runtime: as a new resource allocation is chosen by the *decide* phase, we increase or decrease the number of pCPUs in the pool and pin the domain's vCPUs to these, to increase workload stability. The domain still has the same amount of virtual resources, that *XeMPUPiL* distributed over the maximum number of physical ones available, potentially causing more vCPUs to be time-multiplexed on the same core.

The second tool supported is *xenpm*: this allows to set a maximum and minimum frequency for each pCPU. After a first evaluation, we decide to leave the actuation of the core frequencies out of the *decision* phase, as it may interfere with the actuation made by RAPL.

4. EXPERIMENTAL RESULTS

The goals of our experiments are twofold: (i) we want to show how the metric of performance we propose in this paper behaves when subject to a power limit and (ii) that *XeMPUPiL* is able to maximize that metric given a certain power limit.

Tests have been performed on a system equipped with a 2.8-GHz quad-core Intel Xeon E5-1410 processor (4 hardware threads, TurboBoost and HyperThreading disabled) with 32GB RAM. The system runs the Xen hypervisor version 4.4, with a *paravirtualized* instance of Ubuntu 14.04 as Dom0, pinned on the first core and with 4GB of RAM.

We set up three distinct *paravirtualized* domains, each of those running one of the following four different microbenchmarks, each one representing a different computational class:

1. *NPB3.3 Embarrassingly Parallel (EP)*, a CPU-bound benchmark;
2. *IOzone*, an IO-bound benchmark;
3. *cachebench*, a memory-bound benchmark;
4. *NPB3.3 Block Tri-Diagonal solver (BT)*, a mixed-class benchmark.

EP generates pairs of Gaussian random deviates: this is quite typical of many Monte Carlo simulation applications [4]. IOzone is a filesystem benchmark tool, generating and measuring a variety of different file operations [3]. Cachebench is designed to test memory and cache bandwidth performance [5]. BT is a pseudo application, more specifically a Block Tri-diagonal solver [4]. Experiments have been repeated multiple times, to improve the accuracy of the results.

Our first set of experiments aims to show how the number of IR over a certain time window decreases as the

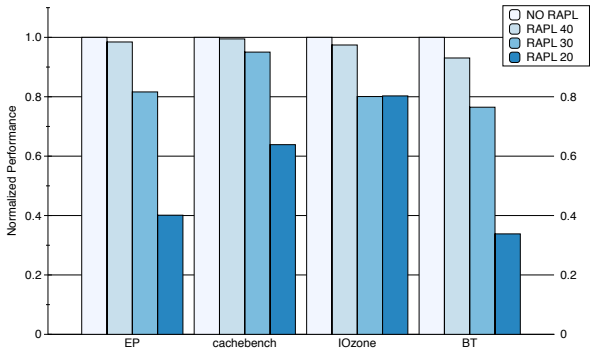

Figure 2: Benchmarks' performance normalized with respect to NO RAPL, under different power caps

power cap becomes stricter. We run each benchmark in a domain with three virtual CPUs assigned and pinned on the three available physical CPUs, to avoid interferences and to maximize resource utilization. We repeated the tests in four different scenarios, namely:

1. `NO RAPL`, as no power limit was set;
2. `RAPL 40`, as a 40W power limit is set (using RAPL);
3. `RAPL 30`, as a 30W power limit is set (using RAPL);
4. `RAPL 20`, as a 20W power limit is set (using RAPL);

We chose 40W and 20W as the maximum and minimum power caps as we observed that, in an idle state, the entire socket consumes around 17W, while the maximum power consumption we reached was around 43W. The comparison between the performance of each single benchmark under different power caps is shown in Figure 2, where the Y-axis reports the performance expressed as the average IR over a time window of 5 seconds. All the results have been normalized with respect to the performance obtained under the `NO RAPL` condition: as expected, the chosen metric is a reasonable indicator of the performance of the application and decreases with a stricter power cap. More in details, with CPU-bound benchmarks (i.e., EP and BT) the difference are greater than in benchmarks where the bottleneck are IO and memory accesses: in these cases, the performance degradation is less significant between different power caps.

The second set of experiments is meant to achieve our second goal: we want to compare the performance of the workloads when *XeMPUPiL* performs its resource allocation in the same scenarios described above, i.e., under a power cap of 40W, 30W and 20W respectively. Results, normalized with respect to the ones obtained in the NO RAPL configuration, are shown in Figures 3, 4 and 5.

XeMPUPiL is able to achieve higher performance under the same power cap in all the scenario and for all the benchmarks: this is due to the decision of assigning in a smart way all the possible domain's vCPUs on fewer

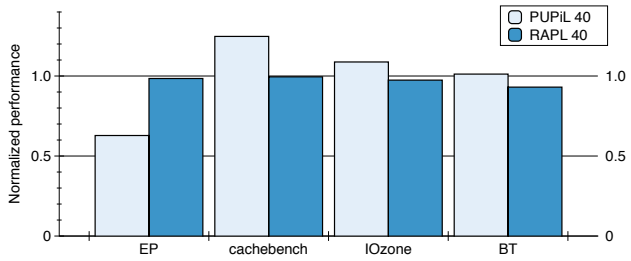

Figure 3: Benchmarks performance normalized with respect to NO RAPL under a cap of 40W, imposed by RAPL and by *XeMPUPiL*

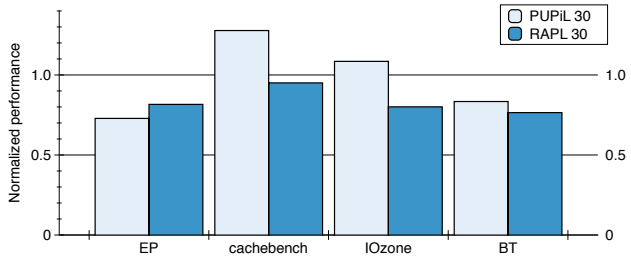

Figure 4: Benchmarks performance normalized with respect to NO RAPL under a cap of 30W, imposed by RAPL and by *XeMPUPiL*

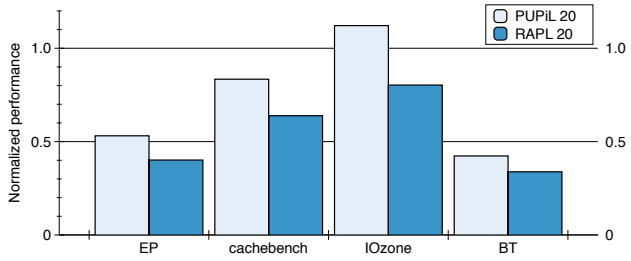

Figure 5: Benchmarks performance normalized with respect to NO RAPL under a cap of 20W, imposed by RAPL and by *XeMPUPiL*

pCPUs than the available ones. However an exception to this trend is represented by the EP benchmark, where in any case the performance gets better for the same benchmark with a cap of 20W, as the framework redistributes the virtual resources over just two physical cores, thus obtaining a configuration that is more power efficient.

As mentioned before it is interesting to note how the performance achieved in case of the IO-bound, the memory-bound and the mixed benchmark are even better than the ones achieved by the `NO RAPL` experiment: for IOzone and cachebench, *XeMPUPiL* converged to a configuration with just one core assigned to the domain, while two cores have been assigned to the BT benchmark. These assignments are more power efficient, as they reduce memory and IO contention for non strictly CPU-bound workloads.

5. LIMITATIONS

This paper presents some preliminary results and opportunities towards a performance-aware power capping orchestrator for the Xen hypervisor. A first limitation is related to the infrastructure used during our test, that does not provide a reasonable amount of resources to explore all the potentials of the proposed approach. Then, a richer set of benchmarks needs to be taken into consideration to reproduce some real-world scenarios, as well as conditions of colocation of benchmarks of the same class or of different classes. Finally, the adopted version of Xen (i.e., version 4.4) presents some performance drawbacks when hyper-threading is enabled: a more recent version of the hypervisor needs to be instrumented to produce interesting results with hyper-threading enabled.

6. CONCLUSION AND FUTURE WORK

In this paper, we presented *XeMPUPiL* , a performance-aware power capping orchestrator for the Xen hypervisor. We extended the current implementation of PUPiL [27] to make it work in a virtualized environment based on the Xen hypervisor, without requiring any instrumentation of the guest workloads. The proposed solution exploits the Intel RAPL hardware interface to set a strict limit on the processors' power consumption, while a software-level ODA loop performs an exploration of the available resource allocations, to find the most power efficient for the running workload. We showed how *XeMPUPiL* is able to achieve higher performances under different power caps for almost all the different classes of benchmarks analyzed (e.g., CPU-, memory- and IO-bound ones).

Future work revolves around a more portable and general implementation of the whole framework, as well as the development of a better decision algorithm, to minimize the duration of the *decide* phase: in fact, its duration currently depends on both the size of the time windows considered and the time required by the workload to reach stable performance once the configuration is chosen and applied. Moreover, we want to improve the *observe* phase, digging deeper into the XeMPower tool to weight the Instruction Retired metric on the number of the "clock-ticks" in the observed interval, thus obtaining a Clockticks per Instructions Retired (CPI) metric. Finally, we want to improve the actuation phase, implementing custom fine-grain tools, since the actual CLI provided by Xen allows only a limited set of resources to be tuned.

7. REFERENCES

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

[2] The embedded and automotive team within the xen project. `https://www.xenproject.org/developers/teams/embedded-and-automotive.html`. Accessed: 2016-09-17.

[3] Iozone filesystem benchmark. `http://www.iozone.org`. Accessed: 2016-06-01.

[4] Nas parallel benchmarks. `http://www.nas.nasa.gov/publications/npb.html#url`. Accessed: 2016-06-01.

[5] Openbenchmarking.org. `https://openbenchmarking.org/test/pts/cachebench`. Accessed: 2016-06-01.

[6] The xen project - success stories. `http://www.xenproject.org/users/success-stories.html`. Accessed: 2016-06-01.

[7] I. Ali and N. Meghanathan. Virtual machines and networks-installation, performance study, advantages and virtualization options. *arXiv preprint arXiv:1105.0061*, 2011.

[8] 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.

[9] R. Cochran, C. Hankendi, A. K. Coskun, and S. Reda. Pack & cap: adaptive dvfs and thread packing under power caps. In *International Symposium on Microarchitecture (MICRO)*, 2011.

[10] 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.

[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] 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.

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

[14] T. Horvath, T. Abdelzaher, K. Skadron, and X. Liu. Dynamic voltage scaling in multitier web servers with end-to-end delay control. In *Computers, IEEE Transactions*. IEEE, 2007.

[15] Intel. Flexible, low power microservers for lightweight scale-out workloads. Technical report, White paper, Intel Corporation, 2013.

[16] J. M. Kaplan, W. Forrest, and N. Kindler. Revolutionizing data center energy efficiency. Technical report, Technical report, McKinsey & Company, 2008.

[17] P. Karger and A. Herbert. An augmented capability architecture to support lattice security and traceability of access. In *IEEE Symposium on Security and Privacy*, 1984.

[18] D. H. K. Kim, C. Imes, and H. Hoffmann. Racing and pacing to idle: Theoretical and empirical analysis of energy optimization heuristics. In *International Conference on Architectural Support*

for Programming Languages and Operating Systems (ASPLOS), 2013.

[19] M. Kim, M. O. Stehr, C. Talcott, N. Dutt, and N. Venkatasubramanian. xtune: A formal methodology for crosslayer tuning of mobile embedded systems. In *ACM Trans. Embed. Comput. Syst. 11.4*. ACM, 2013.

[20] R. Kumar and S. Charu. Comparison between cloud computing, grid computing, cluster computing and virtualization.

[21] D. Meisner, C. M. Sadler, L. A. Barroso, W. Weber, and T. F. Wenisch. Power management of online data intensive services. In *International Symposium on Computer Architecture (ISCA)*.

[22] 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 *International Parallel & Distributed Processing Symposium (IPDPS)*. IEEE, 2005.

[23] A. A. Semnanian, J. Pham, B. Englert, and X. Wu. Virtualization technology and its impact on computer hardware architecture. In *Information Technology: New Generations (ITNG), 2011 Eighth International Conference on*, pages 719–724. IEEE, 2011.

[24] 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.

[25] A. Vasan, A. Sivasubramaniam, V. Shimpi, T. Sivabalan, and R. Subbiah. Worth their watts?-an empirical study of datacenter servers. In *High Performance Computer Architecture (HPCA), 2010 IEEE 16th International Symposium on*, pages 1–10. IEEE, 2010.

[26] 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.

[27] H. Zhang and H. Hoffmann. Maximizing performance under a power cap: A comparison of hardware, software, and hybrid techniques. In *International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS)*, 2016.