Power Budgeting of Big Data Applications in Container-based Clusters

Jonatan Enes*, Guillaume Fieni†, Roberto R. Expósito*, Romain Rouvoy†§, Juan Touriño*
*Universidade da Coruña, CITIC, Spain, †University of Lille / Inria, France, §IUF, France
*Email: {jonatan.enes, roberto.rey.exposito, juan}@udc.es, †Email: {guillaume.fieni, romain.rouvoy}@univ-lille.fr

***Abstract*—Energy consumption is currently highly regarded on computing systems for many reasons, such as improving the environmental impact and reducing operational costs considering the rising price of energy. Previous works have analysed how to improve energy efficiency from the entire infrastructure down to individual computing instances (*e.g.*, virtual machines). However, the research is more scarce when it comes to controlling energy consumption, specially in real time and at the software level. This paper presents a platform that manages a power budget to cap the energy consumed from users to applications and down to individual instances. Using containers as virtualization technology, the energy limitation is implemented thanks to the platform's ability to monitor container energy consumption and dynamically adjust its CPU resources via vertical scaling as required. Representative Big Data applications have been deployed on the platform to prove the feasibility of this approach for energy control, showing that it is possible to distribute and enforce a power budget among users and applications.**

***Index Terms*—Energy consumption, Big Data, Container-based virtualization, Power budget, Resource scaling**

I. Introduction

Energy efficiency is currently one of the most studied research topics across computer systems for its importance as part of the movement towards a more environmentally sustainable future. With well-established computing paradigms, such as the Cloud [1], or the rise of computing-intensive fields, such as Big Data [2], the energy consumed by users and applications is becoming of prime importance for system administrators and service providers.

However, most of the published literature focuses on the energy consumed as a whole by entire systems [3], individual physical hosts [4], [5] or, at the most, by Virtual Machines (VM) [6], as these are the most common means of virtualization. The aim of this paper is to contribute to moving the research forward when it comes to energy monitoring and management in large infrastructures [7], [8], such as Big Data clusters, while going over some of the least covered scenarios. To do so, we study, address and combine fine-grain software-based energy monitoring with an energy capping capability applied to OS containers. More precisely, our main objective is to be able to manage energy as a first-class resource that can be accounted, restricted and shared among applications or users, in the same way as CPU or memory. With this goal in mind, several technologies and tools are integrated and deployed to implement a platform on which Big Data applications can be executed in container clusters, allowing users to set a power budget on their applications and have it enforced at any moment, in a dynamic way and in real time. The key contributions of this paper can be summarized as:

1) We introduce a novel platform to manage energy as any other system resource, which can be distributed among containers, applications and users in real time;
2) We analyse two representative use cases of the platform running Big Data workloads and showing how energy can be distributed dynamically in container-based clusters.

II. Background

In this work, energy monitoring focuses on CPU exclusively and, more precisely, on individual containers. This decision is not only backed by technical limitations, but also by design choices. First, considering that we rely on software-level energy metrics, energy usages that are out of the actual server chassis (*e.g.*, cooling, rack) are discarded to avoid any physical hardware deployment. Second, any energy study that uses software-level monitoring exclusively, particularly for virtualization technologies like containers [9], [10], is limited to CPU and memory energy metrics as these are typically provided by RAPL directly or inferred via lower-level processor information (*e.g.*, Linux *perf* tool). This fact leaves out devices, such as disks and network cards. Finally, from the CPU and memory energy metrics that are available, only CPU is used as per design our platform scales CPU resources according to CPU usage, without taking into account memory.

Regarding the motivation for using containers as the means of virtualization, it has to be noted that besides their increasing popularity for hosting a wide range of applications and services, the most popular container engines (*e.g.*, Docker [11], LXC [12]) rely on cgroups as the underlying means to manage the resources given to containers. This technical feature makes it possible to change their resource limits in real time by setting the appropriate values within the cgroups file system. Our platform relies on this feature to implement a form of throttling to limit the energy consumed by the containers. Such throttling is implemented via a precise, fine-grain control of the CPU shares that containers have at any moment, taking advantage of the ability to do so with cgroups. However, some limitations have to be taken into account when deploying the user's applications using containers: 1) energy can only be limited for those applications (or parts of) that are containerized; 2) containers have to be private and user attached; and 3) applications should have some flexibility regarding the number of cores presented to them, as they may vary along time.

Finally, some concepts regarding energy require to be defined as they will be used later on. First, the term *power budget* is used to express a given power limit that should not be surpassed, which can be applied to the entire platform, users and applications (as a set of containers). Second, the concept of *energy proportionality* [13], [14] is used to explain the behaviour of current CPUs when it comes to the energy consumed compared to the work being performed. Such behaviour can be expressed as how the relationship between both variables evolves according to the CPU load. On an ideal CPU, such relationship would be constant and linear, that is, the energy consumed by the CPU in idle state is zero, half the maximum power at half load and the maximum when usage is the highest. Unfortunately, real CPUs behave quite differently, causing an inescapable impact on the experiments, as further discussed in Section V. With the currently available hardware, not only the energy consumed is not zero when the CPU is idle, thus having an *idle energy consumption*, but also its evolution according to the load is logarithmic instead of linear [15]. So, we define the *energy efficiency ratio* as the amount of CPU shares used according to the energy consumed in a given time window.

III. Related Work

Previous works have studied different energy-related topics on Cloud environments, using VMs or containers. These topics include energy monitoring, management or ultimately capping on systems, from individual applications on single processors to entire workflows across data centers.

Regarding energy monitoring, a few tools can report on energy metrics from running containers, such as DEEP-mon [9] and PowerAPI [16], [17]. Both solutions rely on low-level operating system utilities (*e.g.*, *perf*) that enable precise control over events triggered by individual processes or threads. These events, in conjunction with other system metrics, are strongly correlated and thus can be used to formulate a power estimation of a container. In addition, these solutions impose a near-negligible overhead. Related to this topic, it is interesting to point out a study of the energy consumption overhead that container-based virtualization imposes on the applications [18]. In this work, the authors run several workloads in and out of Docker containers, concluding that they pose a small overhead both in runtime and energy consumption, the latter being mostly related to the former.

There are other works that present frameworks and platforms with a similar objective to the one proposed in this paper, which are worth describing in more detail:

- DockerCap [10] is a software-level power capping framework for Docker containers. This framework is similar to our platform in that it can enforce an energy limit to be respected. Nonetheless, our work goes further by considering not only individual containers, but also clusters of them across several hosts, while also targeting much more complex workloads and higher-level abstractions such as multi-container applications and users. We also present the energy overheads that inevitably arise due to the longer execution times when restrictions are applied, giving some insight into how these overheads vary according to the underlying CPUs' energy efficiency and proportionality. This is further exposed with the larger range of power caps we use (DockerCap goes from 20W to 40W, while we go up to 1500W);
- Nornir is described in [19], [20] as a self-adapting framework capable of enforcing energy constraints dynamically by using mechanisms such as DVFS or core reallocation. Most interestingly, this work uses a feedback loop similar to the one used for DockerCap and our platform, backing it as a good strategy for real-time energy enforcement. However, Nornir focuses on parallel applications running on shared-memory systems, leaving out any virtualization technology.

To summarize and put the proposed platform in context with the previous related work, it has to be noted that our main objective is to define a power budget that can be enforced on users, applications and containers using vertical scaling of CPU resources guided by specific policies.

IV. Power Budget Platform

Our main goal is to provide users with a container-based platform where energy consumption can be monitored and, to a certain limit, enforced. The overall architecture of the proposed platform is described in Section IV-A. The control of the energy consumed by the containers via CPU throttling is thoroughly described in Sections IV-B and IV-C.

A. Architecture Overview

Considering the container as the basic infrastructure unit that can be managed within the platform, the following entities are defined to create richer and more flexible scenarios: *containers*, *applications* and *users*. So, the proposed platform is organized as a three-level hierarchy, as depicted in Figure 1, showing the aforementioned entities from bottom to top. At the lowest level, multiple containers can be grouped to form an application. At the intermediate level, an application acts as an abstract structure that adds up all the containers' CPU and energy metrics and, likewise, allows setting a power budget to be respected by the grouped containers as a whole. Users would be placed at the highest level, with applications attached to them. Similar to applications, each user has an accounting for the grouped energy and CPU and a power budget, which is enforced down to the user's applications through their own budgets. Note that containers do not have any specific energy limit, as power control is only considered at the user and application levels. As containers are treated as black boxes that are part of a more complex structure—*i.e.*, an application—there is no point in limiting the energy of specific containers.

With the above hierarchy laid out, the procedure to propagate a global power budget down to the CPU limits of the containers can be described, from top to bottom, in four steps (numbered in Figure 1): 1) the global power budget is dynamically divided among the users according to their aggregated energy consumption; 2) at the user level, the power

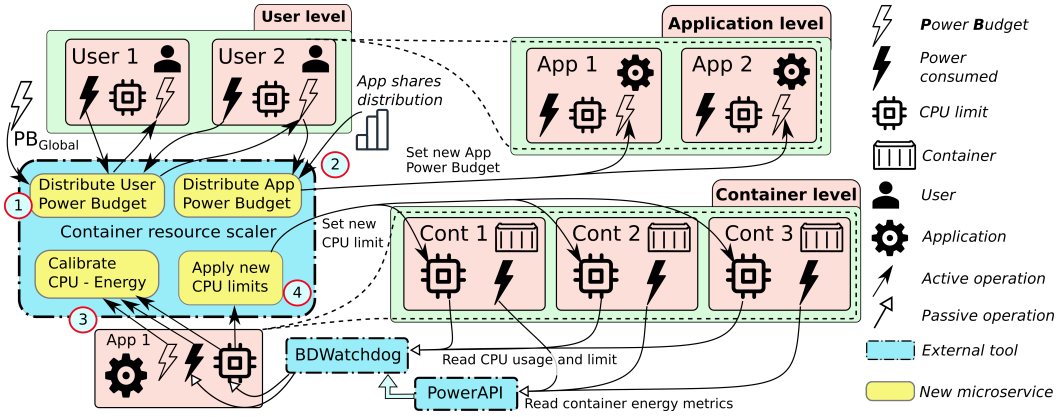

Fig. 1: High-level overview of the platform architecture

budget is statically distributed among all its applications by the container resource scaler tool [21] according to a ratio configured by the user, thus setting a power budget for each application; 3) at the application level, the power budget is compared to the actual energy that is being consumed at the moment to determine if any operation is needed to be carried out. If that is the case, a request is created with the number of CPU shares to be increased or decreased; and 4) if such request is created, it is applied across all the containers by adapting their CPU limits using cgroups. The first two steps defined are fairly simple, as they merely distribute a power budget dynamically (step 1) or statically according to a set of ratios (step 2). However, steps 3 and 4 are more complex and are further described in Sections IV-B and IV-C, respectively. All these steps, referred to as active operations, are carried out in order as needed (see Figure 1).

Regarding the external tools chosen to implement this platform, several had to be adapted for the task in hand. For the core feature of scaling the containers' CPU, a framework capable of creating a serverless environment [21] was chosen as it readily supports containers and has proved to be responsive enough in order to minimize overheads. For resource monitoring, two different tools were used for measuring CPU and energy metrics, referred to as passive operations in Figure 1. On the one hand, POWERAPI [16], [17] was used for energy monitoring as it is capable of reporting container energy usage in real time with high accuracy. On the other hand, BDWATCHDOG [22] was chosen for CPU monitoring and to store the energy metrics as provided by POWERAPI.

B. Power Budgeting Policy

The CPU usage and energy consumption of applications play a key role when making the decision to change the CPU limits during step 3. A first and simple policy would be one that only takes into account energy consumption and that scales the CPU according to whether it surpasses or not a given threshold. However, this policy can easily cause instability as the CPU patterns of Big Data applications can be unpredictable at times considering that they generally go through multiple processing stages that demand different amount of resources. To mitigate this issue, CPU usage is also taken into account to define a richer and more robust policy, as shown in Table I.

To illustrate the behaviour of the platform when deciding if an action has to be taken, we use Figure 2 together with Table I. This figure reports on the time series for the aggregated CPU usage and energy consumption of a synthetic workload used as a representative example. At the beginning of the execution, CPU and energy remain low from second 0 to around 70, so there is no need for any action (State 3 in Table I). Then, from second 70 to around 220, there is a sharp increase in CPU usage and, in turn, in energy consumed. This sudden CPU spike causes the energy consumed to be above the initial power budget (750 J/s), thus prompting the platform to act and scale down the CPU shares to reduce energy consumption (State 2). This action decreases CPU shares from $12,000$—*i.e.*, 120 cores—to about $8,000$, which causes energy to be under the cap from second 220 to around 300. Note that, around second 220 to 240, the CPU limit is the exact one to cause energy consumption to be just below the cap, which can be seen as a perfectly controlled and stable state. Both CPU and energy remain low from second 250 to 300, so no changes are needed and the platform goes back to State 3, with the difference that the CPU limit is now more realistic according to the power budget. At second 300, the budget is increased from 750 J/s to $1,000$ J/s while at the same time the CPU usage rises significantly until it becomes a bottleneck, causing the platform to transition to State 4. This state (the opposite of State 2) implies that the application is using close to all of its CPU shares but, at the same time, is not fully using its power budget. To act accordingly, the CPU limit is progressively raised up to $12,000$ shares until the energy consumed is near but below the budget (around second 460). From second 460 to around 560, the platform enters State 1, where, although the CPU is near a bottleneck, energy consumption is really close to the limit and thus no action is taken. Finally, CPU usage and energy consumption decrease after second 560, so the platform switches to State 3, but keeping the CPU limit at $12,000$ shares.

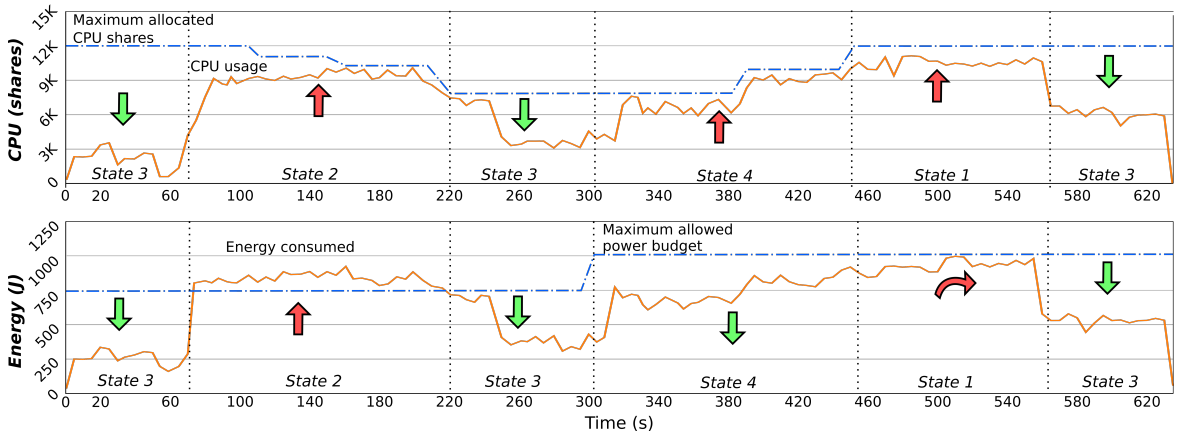

Fig. 2: Time series for the CPU usage (top) and energy consumption (bottom) of the application

TABLE I: Policy decision criteria (PB: power budget)

State	CPU Usage	Energy Consum.	Decision
1	-	In PB	Keep CPU
2	-	Above PB	Decrease CPU
3	Low/Medium	Under PB	Keep CPU
4	High/Bottleneck	Under PB	Increase CPU

TABLE II: Host hardware configuration

CPU	2x Intel Xeon Gold 6126 @2.60GHz [24 cores]
Memory	192 GiB DDR4
Disks	2x 480 GiB SSD SATA Intel (OS, root volumes) 4x 4 TiB HDD SAS (data)
Network	2x10 Gbps

C. CPU Scaling Policy

A request to scale up/down the CPU shares of an application, which groups multiple containers, must be translated to individual requests for each container during step 4 of Figure 1. Considering that such request can only be to either decrease or increase the CPU limit to lower or raise energy consumption, respectively, the policy to translate such application requests to container requests can be determined as follows: 1) if the request is to increase the CPU limit, the policy raises it to those containers that have a lower CPU utilization; 2) if the request is to decrease the CPU limit, the policy reduces it to those containers with the highest CPU usage, considering that this is the fastest way of decreasing the overall energy consumed by the application.

V. Big Data Use Cases

To analyse the efficiency of our energy control policy, we consider a realistic Big Data setup with two experiments that represent use cases from different domains. These experiments are evaluated to show the feasibility and effectiveness of the proposed platform with different objectives to prove: 1) static power budgeting using a machine learning workload; and 2) user-level budgeting using a streaming application. It is worth noting that although the deployed applications use Big Data technologies, they do not rely heavily on I/O but rather on in-memory processing and thus they are mainly CPU-bound. The characteristics of the hardware used and the configuration of the platform are detailed in Section V-A, whereas the experiments are described in Section V-B.

A. Hardware & Platform Configuration

To deploy the experimental testbeds, several nodes from the Grid'5000 infrastructure [23] have been used, referred to from now on as 'hosts'. The experiments are carried out on LXC-based container clusters deployed using physical hosts with the hardware characteristics described in Table II. Each LXC container runs Ubuntu 18.04 LTS and Java JDK 1.8.0_212, and it is deployed with a maximum of 600 CPU shares and 45 GiB of memory. Four containers per host are used, as this is a good ratio to distribute the pool of host resources across a set of containers to avoid either a group of containers with few but fat instances, or many but thin instances. Regarding the tools, BDWatchdog and PowerAPI have a monitoring polling frequency of 5 seconds, while the container resource scaler tool uses a policy to scale CPU resources using energy as input metric and works with 40-second time windows.

B. Experimental Results

The results for the two experiments are presented next. All the plots show the aggregated CPU usage (shares) and energy consumption (Joules) during the execution of the experiments. Although the plots only show the most representative execution for each experiment, the metrics provided are obtained from the average of a minimum of 5 executions.

1) Static Power Budgeting: In this experiment, a single user deploys one application running KMeans, an iterative clustering technique, using Spark 2.3.0 [24]. The input dataset contains 20 million samples and the algorithm performs 10 iterations using 30 clusters. This workload was chosen due to its iterative behaviour and variable CPU and energy patterns, with periods of high and low load. This is useful to prove that the platform is able to effectively throttle the energy consumed by the applications in real time and with no previous information about the workloads. To run the experiment, 8 hosts are used to deploy a 32-container cluster with Hadoop

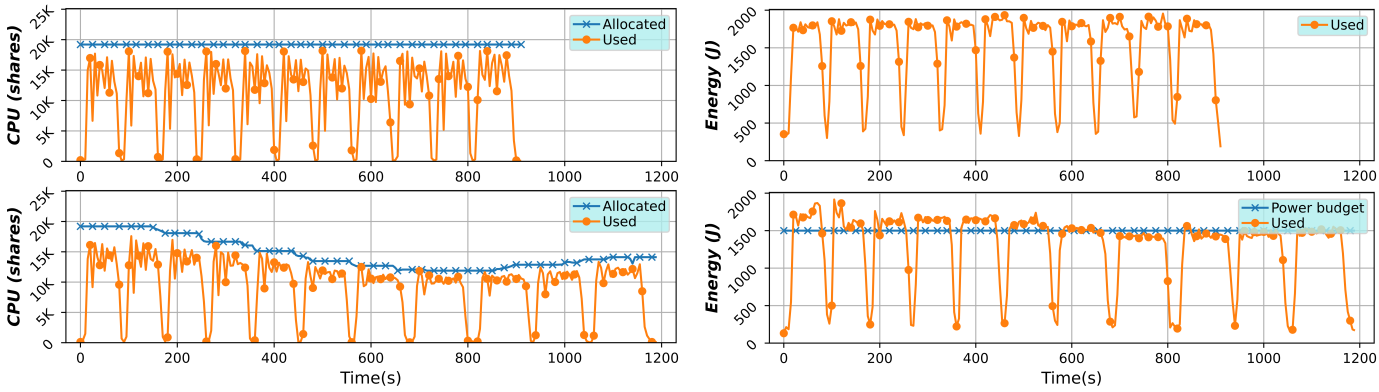

Fig. 3: CPU usage and energy consumption for KMEANS (top plots: baseline scenario, bottom plots: capped scenario)

2.9.0 [25]. The *Big Data Evaluator* (BDEv) tool [26] is used to set up Hadoop and to run the workloads automatically.

In order to better see the platform in action, two scenarios are compared for this experiment, a scenario where the workload is executed without any power budgeting or interference overall from the platform, referred to as *baseline*, and a second one where capping is applied, named as *capped*. Figure 3 shows CPU usage (left plots) and energy consumption (right plots) for KMEANS. It can be seen how after setting a limit of $1,500$ J/s for the capped scenario, energy consumption is brought down under control after enough time passes (around second 600), although it has to be noted that the workload began the execution with full CPU resources available. The power budget set to this workload aims to bring the peaks of energy consumption of up to $1,800$ J/s in the baseline scenario down to under $1,500$.

It is interesting to note that the CPU limit reached at second 700 (around $12,500$ shares) appears to be lower than the one strictly required. This in turn causes energy consumption to drop below the maximum between seconds 700 and 800 (8th iteration of the algorithm). During this iteration, the platform switches to State 4 (see Table I), where the CPU usage has a bottleneck and simultaneously the energy consumed is below the cap. The platform tackles this issue and transitions to the really stable State 1 by slightly increasing the CPU resources, staying in that state for the remaining iterations.

As a result of the limit imposed, the average energy consumption per second is reduced from $1,450$ J/s for the baseline scenario to $1,320$ J/s (-9%) for the capped one. Regarding total consumption, it is $1,356$ and $1,638$ kJ (+21%), with runtimes of 15 and 20 minutes (+33%), respectively. The total CPU usage (calculated as the aggregated number of cores that the experiment used times its runtime) remains the same for both scenarios ($1,783$ core-minutes). The explanation for these metrics lies in the fact that, while the workload is not affected—*i.e.*, it does the same operations—, its runtime obviously increases due to the CPU restrictions. In turn, the longer runtime increases the total energy consumption due to the lack of energy proportionality of the CPU, as mentioned at the end of Section II, which heavily penalizes longer runtimes due to the idle energy consumption.

2) User-level Dynamic Power Budgeting: The second experiment aims to show how a global power budget for the entire infrastructure can be dynamically divided among users according to their needs. While for the first experiment a specific power cap is used (previously acquired knowledge), this one shows a scenario where energy is dynamically distributed. In this experiment, there are two different users with separate and dedicated environments running each one a streaming application deployed across 8 containers, which consists of: 1) a data generator from HiBench 7.0 [27] (1 container); 2) a message broker based on Kafka 2.1.0 [28] (2 containers); and 3) a Spark 2.3.0 cluster running the FixWindow workload (5 containers), also obtained from HiBench. In total, 16 containers are deployed across 4 physical hosts mixing the containers of both users on each one. To simulate varying loads for the users, the application is configured with three stream sizes that can be changed at any time for any user, named large (171 MiB/s), medium (76 MiB/s) and small (38 MiB/s). The streams are processed in 10-second windows.

Figure 4 shows CPU usage and energy consumption for the streaming application executed by both users. For this experiment, three stages are created to emulate different scenarios, named in order of execution (see Table III): 1) Balance, where User 1 incrementally scales up its stream (3 streams, small-medium-large) while User 2 does the same in reverse (large-medium-small); 2) Contention, where both users process a large stream to create a contention scenario; and 3) Efficiency, where User 1 processes 3 consecutive large streams while User 2 first remains idle and then processes two streams of small and large size, respectively.

The objective of these stages is different. On the one hand, Balance and Contention aim to show how a global power budget for the entire infrastructure can be shared and balanced among users and their applications. The difference between both lies in the fact that, while for the Balance stage the budget is enough to accommodate both users, as their CPU and energy requirements are inversely correlated over time, in the Contention stage both users have simultaneously high resource requirements and the budget has to be split. In this

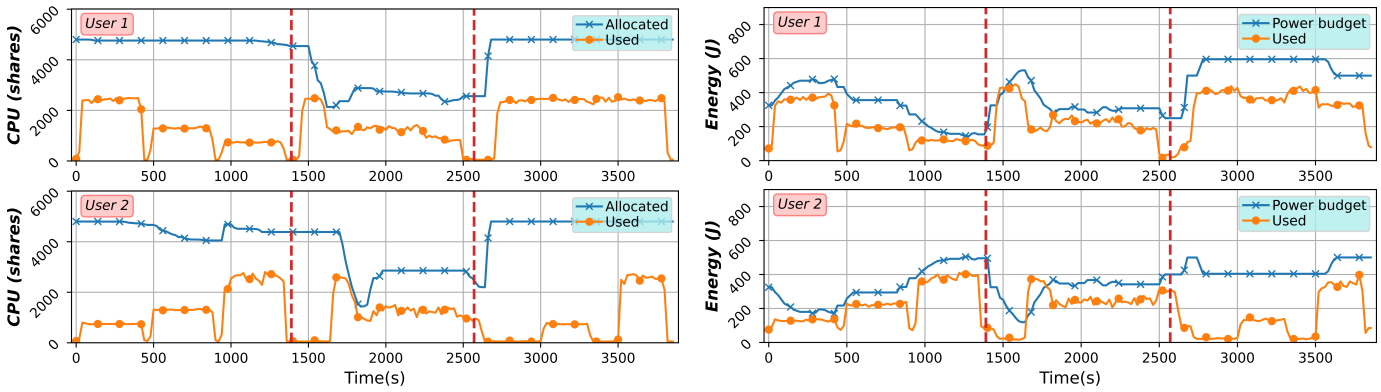

Fig. 4: CPU usage and energy consumption for streaming (stages Balance, Contention and Efficiency split by dashed lines)

TABLE III: Average energy consumption per second (J/s)

	Balance			Contention	Efficiency		
stream	1	2	3		1	2	3
User 1	321	185	117	210	386	324	306
User 2	124	204	323	215	28	119	317

latter case, as the budget is not enough to process two large streams, a scale down process has to be carried out. On the other hand, the Efficiency stage serves to further analyse the variation of the energy efficiency according to the load of the underlying physical host, which unfortunately affects the users even if their processing environments are virtually separated.

As it can be seen in Figure 4, the Balance stage lasts from the beginning to around second $1,400$, with each stream running for 450 seconds. An initial power budget of 650 J/s is shared between both users according to their requirements, shifting a budget of around 475 J/s from User 1 in the first stream to User 2 at the end of the stage. After Balance finishes, the Contention stage begins lasting from second $1,400$ to around $2,600$. In this stage, the same budget (650 J/s) is similarly split between users. However, because this budget is insufficient to cater for both users, their applications are scaled down from an expected average energy consumption of around 320 J/s when processing a large stream (see first stream of User 1 for the Balance stage in Table III) down to 210 J/s. Finally, the Efficiency stage begins at around second $2,600$ and the global power budget is increased from 650 J/s to $1,000$ J/s (in this stage the budget enforcement is not the focus). It can be seen in Table III how the streaming application of User 2 lowers the average energy consumption of User 1: from 386 J/s (User 2 is idle) to 324 (-16%) and 306 (-21%) when User 2 is processing the small and large streams, respectively. This behaviour can be explained by the fact that the higher the underlying host utilization, the higher the energy efficiency—*i.e.*, less amount of energy for the same processing requirements—. Using the energy efficiency ratio as defined at the end of Section II, the values obtained for the Efficiency stage are 1.02, 1.13 (+11%) and 1.24 (+22%) core-minutes/kJ for the idle, small and large streams, respectively. This further proves how the most efficient scenario is the one where the underlying hosts have the CPU usage as high as possible, in this case, when both users are processing a large stream.

VI. Conclusions

In this work, energy was presented as another system resource that can be likewise shared and split across users, applications and containers. Moreover, it can be capped at the software level to ensure that energy consumption is kept below a certain limit. To implement this concept, a power budget platform was created by integrating and extending several tools so that the CPU shares of containers can be scaled down or up in order to either reduce the energy or allow it to raise, respectively. To prove that such energy management can be effectively carried out for Big Data applications as a representative real-world scenario, two experiments were deployed on the platform to expose different use cases.

The experimental results showed that it is possible to transparently enforce a certain power limit without incurring any overhead in terms of the amount of CPU required to complete a task. However, the limit is configurable and can be adapted to the specific needs of users and applications, particularly useful to minimize the total energy consumption and runtime overhead. Furthermore, our platform can manage a power budget that is dynamically distributed between several users as shown with the streaming application. The proposed platform is publicly available at http://bdwatchdog.dec.udc.es/energy.

Acknowledgments

This work was supported by the Ministry of Science and Innovation of Spain (TIN2016-75845-P and PID2019-104184RB-I00, AEI/FEDER/EU, 10.13039/501100011033); and by Xunta de Galicia and FEDER funds (Centro de Investigación de Galicia accreditation 2019-2022, ED431G 2019/01, as well as Consolidation Program of Competitive Reference Groups, ED431C 2017/04). The experiments were carried out using the Grid'5000 testbed, supported by a scientific interest group hosted by Inria and including CNRS, RENATER and several universities as well as other organizations.

REFERENCES

[1] R. Buyya, C. S. Yeo, S. Venugopal, J. Broberg, and I. Brandic, "Cloud computing and emerging IT platforms: vision, hype, and reality for delivering computing as the 5th utility," *Future Generation Computer Systems*, vol. 25, no. 6, pp. 599–616, 2009.

[2] I. A. T. Hashem, I. Yaqoob, N. B. Anuar, S. Mokhtar, A. Gani, and S. U. Khan, "The rise of big data on cloud computing: review and open research issues," *Information Systems*, vol. 47, pp. 98–115, 2015.

[3] M. Xu, A. N. Toosi, and R. Buyya, "iBrownout: an integrated approach for managing energy and brownout in container-based clouds," *IEEE Transactions on Sustainable Computing*, vol. 4, no. 1, pp. 53–66, 2018.

[4] A. Paya and D. C. Marinescu, "Energy-aware load balancing and application scaling for the cloud ecosystem," *IEEE Transactions on Cloud Computing*, vol. 5, no. 1, pp. 15–27, 2015.

[5] G. L. Stavrinides and H. D. Karatza, "An energy-efficient, QoS-aware and cost-effective scheduling approach for real-time workflow applications in cloud computing systems utilizing DVFS and approximate computations," *Future Generation Computer Systems*, vol. 96, pp. 216–226, 2019.

[6] N. Kim, J. Cho, and E. Seo, "Energy-credit scheduler: an energy-aware virtual machine scheduler for cloud systems," *Future Generation Computer Systems*, vol. 32, pp. 128–137, 2014.

[7] F. Almeida, M. D. Assunção, J. Barbosa, V. Blanco, I. Brandic, G. Da Costa, M. F. Dolz, A. C. Elster, M. Jarus, H. D. Karatza *et al.*, "Energy monitoring as an essential building block towards sustainable ultrascale systems," *Sustainable Computing: Informatics and Systems*, vol. 17, pp. 27–42, 2018.

[8] M. E. M. Diouri, M. F. Dolz, O. Glück, L. Lefèvre, P. Alonso, S. Catalán, R. Mayo, and E. S. Quintana-Ortí, "Assessing power monitoring approaches for energy and power analysis of computers," *Sustainable Computing: Informatics and Systems*, vol. 4, no. 2, pp. 68–82, 2014.

[9] R. Brondolin, T. Sardelli, and M. D. Santambrogio, "DEEP-mon: dynamic and energy efficient power monitoring for container-based infrastructures," in *Proceedings of the 32nd IEEE International Parallel and Distributed Processing Symposium Workshops (IPDPSW 2018)*, Vancouver, BC, Canada, 2018, pp. 676–684.

[10] A. Asnaghi, M. Ferroni, and M. D. Santambrogio, "DockerCap: a software-level power capping orchestrator for Docker containers," in *Proceedings of the 14th IEEE International Conference on Embedded and Ubiquitous Computing (EUC 2016)*, Paris, France, 2016, pp. 90–97.

[11] D. Merkel, "Docker: lightweight Linux containers for consistent development and deployment," *Linux Journal*, vol. 239, pp. 76–91, 2014.

[12] D. Bernstein, "Containers and cloud: from LXC to Docker to Kubernetes," *IEEE Cloud Computing*, vol. 1, no. 3, pp. 81–84, 2014.

[13] L. A. Barroso and U. Hölzle, "The case for energy-proportional computing," *Computer*, vol. 40, no. 12, pp. 33–37, 2007.

[14] D. Lo, L. Cheng, R. Govindaraju, L. A. Barroso, and C. Kozyrakis, "Towards energy proportionality for large-scale latency-critical workloads," in *Proceedings of the 41st ACM/IEEE International Symposium on Computer Architecture (ISCA 2014)*, Minneapolis, MN, USA, 2014, pp. 301–312.

[15] R. Sen and D. A. Wood, "Energy-proportional computing: a new definition," *Computer*, vol. 50, no. 8, pp. 26–33, 2017.

[16] G. Fieni, R. Rouvoy, and L. Seinturier, "SmartWatts: Self-calibrating software-defined power meter for containers," in *Proceedings of the 20th IEEE/ACM International Symposium on Cluster, Cloud and Internet Computing (CCGrid 2020)*, Melbourne, Australia, 2020, pp. 479–488.

[17] 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 Tenth European Conference on Computer Systems (EuroSys'15)*, Bordeaux, France, 2015, pp. 14:1–14:14.

[18] E. A. Santos, C. McLean, C. Solinas, and A. Hindle, "How does Docker affect energy consumption? Evaluating workloads in and out of Docker containers," *Journal of Systems and Software*, vol. 146, pp. 14–25, 2018.

[19] D. De Sensi, M. Torquati, and M. Danelutto, "A reconfiguration algorithm for power-aware parallel applications," *ACM Transactions on Architecture and Code Optimization*, vol. 13, no. 4, pp. 43:1–43:25, 2016.

[20] D. De Sensi, T. De Matteis, and M. Danelutto, "Simplifying self-adaptive and power-aware computing with Nornir," *Future Generation Computer Systems*, vol. 87, pp. 136–151, 2018.

[21] J. Enes, R. R. Expósito, and J. Touriño, "Real-time resource scaling platform for big data workloads on serverless environments," *Future Generation Computer Systems*, vol. 105, pp. 361–379, 2020.

[22] ——, "BDWatchdog: real-time monitoring and profiling of big data applications and frameworks," *Future Generation Computer Systems*, vol. 87, pp. 420–437, 2018.

[23] Grid'5000 testbed for experiment-driven research. [Online]. Available: https://www.grid5000.fr

[24] M. Zaharia, R. S. Xin, P. Wendell, T. Das, M. Armbrust, A. Dave, X. Meng, J. Rosen, S. Venkataraman, M. J. Franklin, A. Ghodsi, J. Gonzalez, S. Shenker, and I. Stoica, "Apache Spark: a unified engine for big data processing," *Communications of the ACM*, vol. 59, no. 11, pp. 56–65, 2016.

[25] The Apache Software Foundation. Apache Hadoop. [Online]. Available: http://hadoop.apache.org

[26] J. Veiga, J. Enes, R. R. Expósito, and J. Touriño, "BDEv 3.0: energy efficiency and microarchitectural characterization of big data processing frameworks," *Future Generation Computer Systems*, vol. 86, pp. 565–581, 2018.

[27] S. Huang, J. Huang, J. Dai, T. Xie, and B. Huang, "The HiBench benchmark suite: characterization of the MapReduce-based data analysis," in *Proceedings of the 26th International Conference on Data Engineering Workshops (ICDEW 2010)*, Long Beach, CA, USA, 2010, pp. 41–51.

[28] N. Garg, *Apache Kafka*. Packt Publishing Ltd., 2013.