Workload Management for Power Efficiency in Heterogeneous Data Centers

Pietro Ruiu*, Alberto Scionti*, Joel Nider†, Mike Rapoport†
* Istituto Superiore Mario Boella (ISMB), Torino, Italy
† IBM Research and Development, Haifa Research Lab
E-mails: *{ruiu,scionti}@ismb.it, †{joeln,rapoport}@il.ibm.com

***Abstract*— The cloud computing paradigm has recently emerged as a convenient solution for running different workloads on highly parallel and scalable infrastructures. One major appeal of cloud computing is its capability of abstracting hardware resources and making them easy to use. Conversely, one of the major challenges for cloud providers is the energy efficiency improvement of their infrastructures. Aimed at overcoming this challenge, heterogeneous architectures have started to become part of the standard equipment used in data centers. Despite this effort, heterogeneous systems remain difficult to program and manage, while their effectiveness has been proven only in the HPC domain. Cloud workloads are different in nature and a way to exploit heterogeneity effectively is still lacking. This paper takes a first step towards an effective use of heterogeneous architectures in cloud infrastructures. It presents an in-depth analysis of cloud workloads, highlighting where energy efficiency can be obtained. The microservices paradigm is then presented as a way of intelligently partitioning applications in such a way that different components can take advantage of the heterogeneous hardware, thus providing energy efficiency. Finally, the integration of microservices and heterogeneous architectures, as well as the challenge of managing legacy applications, is presented in the context of the OPERA project.**

Keywords—cloud computing, power efficiency, workload management, microservices, heterogeneous data center.

I. Introduction

Power consumption in data centers is becoming a prime concern for cloud operators. Approximately 1.3% of worldwide electricity goes towards fuelling data centers [1]. While some of the biggest players such as Google report commendable efficiency numbers, the average efficiency of data centers is only in the range of 10-15%. This means that there is a huge waste of computing resources, but also that there is a huge opportunity for savings. If we can reduce the amount of electricity being wasted, not only will we have "greener" data centers, but we can also reduce operating expenses that could save businesses billions of dollars. Where is the waste coming from? Data centers (DCs) are designed to supply the computing needs for a body of consumers without knowing exactly what the demands will be. Consumer demand can change over time, thus the DC must be able to react to these changes, otherwise prolonged delays may break service level agreements (SLAs) that are in place with the customers. With that in mind, the DC infrastructure is designed to handle the predicted peak demand, with little regard for the times at which the demand is not at its peak. This causes a vast over-provisioning of resources, which is not only wasteful in terms of computing power, but also increases the physical footprint of the data center, and increases cooling and maintenance costs.

Various physical techniques have been employed to increase energy efficiency at several levels: inside the servers, at the rack level, and at the DC infrastructure level [2]. Even though energy-aware scheduling algorithms have been proposed to further improve energy efficiency [3], reported average efficiency numbers remain low [4]. Heterogeneity has been recognized as a way to increase performance and reduce power consumption at the same time. It is based on the adoption of hardware which is specialized for a specific task. A heterogeneous system can be composed of hardware with the same instruction set architecture (ISA) but a different micro-architecture (e.g., ARM big-LITTLE and Intel Xeon PHI architectures), or completely different instruction sets. However, such a difference creates difficulties for programmers that are forced to integrate different programming models within a single application. In the HPC domain this problem was somewhat mitigated where applications are monolithic and could be optimized for the different hardware components. However, this is very costly in terms of programming time and programming expertise, since very few domain experts are able to take full advantage of their computing hardware. Conversely, in the cloud computing domain it remains an open challenge.

In this paper, we look at current trends of cloud applications, and consider how to guide development of energy-aware applications. If it is desired to run them on heterogeneous hardware, the correct development tools must be in place so that the responsibility of efficient execution does not fall primarily on the application developer. One of the necessary components of a power-aware DC infrastructure is a real-time workload management tool that continuously monitors server performance, and can take corrective action to ensure minimal energy consumption. The first step in designing such a tool is to understand the nature of the workloads that run on the system, and to be able to classify them in a format that an automated tool can use. We believe that partitioning the workload into independent blocks (possibly optimized for a specific hardware component), and giving the workload manager the capability of deploying them on the most well suited hardware component allows to improve DC efficiency. In order to analyze the nature of the workloads in the cloud computing domain, we investigate the differences in managing workloads in the three predominant cloud models in use today, namely Infrastructure-as-a-Service (IaaS), Platform-as-a-Service (PaaS), and Software-as-a-Service (SaaS). From the results of our analysis, we can advocate that the SaaS model affords the most opportunity for power savings.

The rest of the paper is organized as follows. Section II summarizes research works that aim at improving power/energy efficiency of data centers by acting at different levels of the system (i.e., exploiting hardware features, heterogeneity, etc.). Section III provides an in-depth analysis of the workloads belonging to the cloud computing domain. The analysis takes into consideration the different cloud models we encounter today (i.e., IaaS, PaaS, and SaaS), and for each of them it provides an evaluation of the potential energy savings achievable. Section IV introduces the microservice architecture paradigm, that will be exploited in the OPERA project (section V). The project is described in terms of adopted hardware architecture and its integration with the microservice architectural pattern. The challenge of integrating legacy applications is described as well. Finally, section VI summarizes the contribution of the work.

II. Data center power efficiency

The electricity delivered to a DC infrastructure is distributed among the IT equipment, cooling system, and other non-IT facilities required to maintain the DC operations, and some waste is inherent due to inefficiencies in the power distribution network. Green Grid created the power utilization effectiveness (PUE) metric [5] as a basis for comparing the energy efficiency of such infrastructures. PUE is determined by dividing the amount of power entering the infrastructure by the power used by the IT equipment within it. PUE optimization may provide substantial power savings [6], however optimizing PUE is not enough because energy delivered to the IT equipment is not completely utilized. A fraction is wasted by the IT equipment power supplies, voltage regulator modules, and cooling fans inside servers and switches. The largest component of DC power inefficiency is server underutilization. According to Google [6], in a hyper-scale cloud infrastructure the percentage of server utilization can vary from 10% to 50%. Among the factors that contribute to low server utilization there is vast over-provisioning of IT resources along with a large deployment of often unused virtual machines. There are a lot of works that address the problem of increasing power efficiency of the servers, taking into consideration different usage scenarios. The research covers different areas, ranging from high-level approaches for workload scheduling to CPU power management techniques.

A. Improving efficiency through heterogeneity

Heterogeneity has been recognized as a viable solution to improve performance and reduce power consumption at the same time [7]. At its basis there are hardware accelerators, such as GPGPUs and FPGAs, with their own specific programming models. Although from an architectural standpoint accelerators are able to process more instructions and data compared to general purpose CPUs, from a technology viewpoint they still need the use of power reduction techniques such as dynamic voltage and frequency scaling (DVFS), adaptive voltage and frequency scaling (AVFS), and partial reconfiguration for FPGAs [8] to keep power consumption under control. The full exploitation of such hardware accelerators in cloud computing environments is still not possible, given the difficulties in programming such systems and making them easily accessible in a virtualized environment. Several research works propose (semi-)automated approaches to offload computations on heterogeneous hardware components. Durelli et al. present the SAVE project [9], that aims at providing a solution for the efficient exploitation of specialized computing resources of a heterogeneous system. By leveraging virtualization of underlying hardware, making the operating system aware of the available resources, and integrating an adaptive resource manager, applications can be run efficiently on the heterogeneous system. Although these concepts are general and may be applicable in the cloud computing domain, the project only demonstrated feasibility for the HPC domain.

Delimitrou and Kozyrakis [10] address heterogeneity in modern DC infrastructures and present Paragon, an interference and heterogeneity aware workload scheduler. The central feature of the Paragon is its classification engine that allows fast and accurate classification of an application, based on two parameters: heterogeneity and interference. This classification allows selecting best server configuration for the incoming workload as well as co-locating of newly arrived workloads with ones that will cause the least interference with each other. Andersen [11] introduced the concept of 'wimpy' nodes for implementation of data-intensive distributed storage. The wimpy server node is based on a low-power embedded processor which is more power efficient that high-end server CPUs. In embarassingly parallel applications, the ability to highly distribute the workload among several wimpy nodes allows the system to achieve high throughput with low power consumption. Other works, such as [12], propose heterogeneous architectures that combine high-performance and low-power servers in order to achieve better overall energy proportionality and energy efficiency.

B. Improving efficiency at the infrastructure level

DC infrastructures can be made energy/power aware by exploiting the interaction of various components of the software stack (i.e., operating system, middleware, virtual machines, etc.) with the underlying hardware. The Muse system [13] proposes consolidation of workloads running inside FreeBSD resource containers allowing full servers to be powered up/down as needed. The work addresses the management of homogeneous resources by using 'bids' and 'penalties' that represent costs associated with the resource usage, SLA violation, and QoS degradation. The Muse system maintains a set of active servers and adjusts the server pool size to optimize for best energy utilization. When a server no longer has any active workloads, it is put into a low power state (hibernated) and eventually shut down. The system only attempts to optimize according to the CPU load and does not take into account DRAM, disks and network cards. In addition, the authors note that their algorithm becomes expensive with changing workloads. Nathuji et al. [14] present power management techniques in a virtualized environment. In addition to virtual machine (VM) consolidation and use of hardware active low power states, they proposed a new concept of soft resource scaling. When a VM decides to transition into a low power state, the underlying virtual machine monitor (VMM) allocates fewer hardware resources to that VM. The approach relies on the guest's ability to effectively manage its power states and therefore if a guest operating system has no advanced power management policies very few savings are available. Both of these approaches are limited in reducing the power consumption of the system, by acting only on the processor states. In fact, although processor DVFS significantly improves energy proportionality of the CPU, the other components of the system (i.e., DRAM, disks, network cards, fans) remain active and their power consumption may reach as much as 75%. PowerNap [15] addresses the lack of energy proportionality in commodity servers and waste of idle-energy. In the PowerNap approach, the entire system transitions rapidly between high-performance active state and a minimal-power nap state, depending on presence or absence of load (also known as *race to idle* [16], [17]). However, this technique is appropriate for services that can tolerate relatively large variations in response latency. Beloglazov's work [18] presents an architectural framework for managing power efficiency in IaaS cloud platforms. The framework is mainly based on algorithms for energy efficient VM placement, as well as on the consolidation of the servers using dynamic migration of VMs to achieve the most power-efficient operation of the data center without significant violation of any SLA. Conversely, a SLA violation is treated like a penalty and as such it is integrated into the placement and migration algorithms. Although this approach is suitable for the general case of IaaS clouds, it relies only on CPU and memory utilization metrics. Moreover, it does not target the other cloud service models, such as PaaS or SaaS, where the cloud management system has a better estimation of the QoS and thus may have more chances to better use resources from an energy viewpoint.

C. Improving efficiency at the application level

As Meisner et al. claim in [19], the online data-intensive (OLDI) services are very sensitive to request-response latencies and hence only low-power active states should be utilized to achieve power savings without violating service level objectives (SLO). The PEGASUS system [20] goes in this direction, and aims to improve power efficiency of latency-sensitive, data-intensive services. The idea is to adjust server performance so that the OLDI workload 'barely meets' its SLO goal. The PEGASUS uses request latency monitoring and SLO targets as inputs of a feedback-based controller that orchestrates server power management systems. Bubble-Up [21] and its successor Bubble-Flux [22] attack low server utilization in warehouse-scale computers. The Bubble-Up system presents a method for improving application co-location for latency-sensitive and batch workloads. The latency-sensitive applications are profiled to obtain estimate on the impact of their co-location with a batch application. This data is then used for the application placement in order to improve server utilization, and without sacrificing the QoS of latency-sensitive workload. The Bubble-Flux advances the concept introduced by Bubble-Up by allowing dynamic selection of co-located workloads and thus eliminating the need for a preprocessing phase.

III. Workload Analysis

Workload classification is not a trivial problem, and is not always successful [23]. Workloads that run in cloud DCs are similar to those for grid computing and high-performance computing (HPC) systems, albeit there are some important differences. At the highest level, workloads can be divided into two groups. *Interactive jobs* (such as OLDI) where a user is waiting for a response in real-time, and *batch jobs* (such as traditional MapReduce) which are generally longer running, with soft expectations for completion time. From this viewpoint, job distribution in cloud computing systems shows a skew towards interactive jobs that is not present in grid or HPC domains. Online shopping, online banking, and real-time analytics are a few of the most popular cloud workload types. That means that cloud workloads are generally sensitive to completion time. On the other hand, batch workloads, such as Spark, running on public clouds and that traditionally would be considered HPC workloads, constitute a small fraction of cloud workloads.

The sensitivity of cloud jobs to completion time means that cloud providers need to be able to provide dedicated resources (CPU, memory, I/O devices, etc.) to their tenants which are always available to handle the demand on the application. However, dedicated resources are expensive, and not all tenants are willing to pay a high price. This is expressed as a trade-off between resource availability and price, which resulted in the creation of the 'spot market' in popular IaaS clouds such as Amazon AWS. In the spot market, the compute resources are priced in a free market environment, driven by demand. A tenant can decide how much a resource is worth, and choose whether or not to buy a resource. This is a short-term lease, since the price of the resource may exceed the price the tenant is willing to pay at any time, and the resource will be freed. The spot market system gives cloud operators some control over resource utilization, and by lowering the price during times of low server utilization, they can entice certain tenants to purchase more compute resources. While spot markets may give some control to recover costs of data centers (DCs), it does not help with power efficiency. Dedicating resources to a tenant (by either a long-term lease or short-term lease) does not ensure that the tenant will take advantage of the resource availability to the fullest extent possible. These dedicated resources can be under-utilized leading to inefficiencies beyond the control of the cloud operator.

A. Energy-aware workload consolidation

In general, IaaS clouds are relatively limited in their ability of optimizing energy efficiency. Monitoring VM efficiency is limited to what the hypervisor can measure from outside the VM. The metrics available to estimate server utilization are CPU load, memory load, and the I/O traffic statistics. To gain more insight, the hypervisor would require the cooperation of the guest OS, or to make some assumptions about the operation of the guest OS. These black-box metrics do not necessarily correlate with the amount of useful work performed by the VM, and thus we can only make best-effort estimations about VM load and efficiency, making it difficult to predict or react to changes. Additionally, the number of tools available that can be used to improve efficiency in the context of IaaS cloud model is quite limited. Practically, the only tool available is VM migration, which bears its own costs due to memory pre-copying and cache warm-up [24].

On the other hand, PaaS model has more opportunities for performing energy-aware workload consolidation. In addition to the metrics available in the IaaS model, the core services of PaaS clouds may provide additional information, like, for instance, the amount of requests per second arriving to a cluster

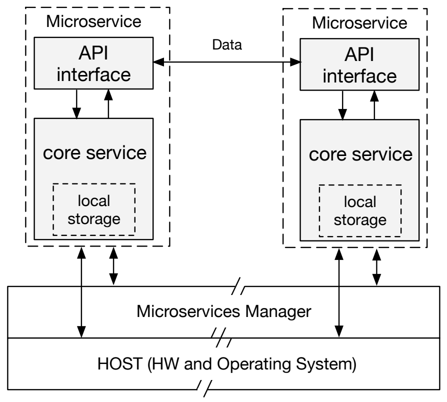

Fig. 1. General microservices architecture. A microservice manager runs on top of the host infrastructure and is responsible for the deployment of microservices. These are isolated entities with their own local storage that export a standard HTTP REST API interface.

of web servers. This information may be used by the cloud orchestration software to make better decisions about workload placement.

The ability to optimize power efficiency in the case of SaaS cloud model is even more prominent. The application running as a service in distributed environment have precise knowledge of its load and may create very accurate prediction of future resource requirements. This allows adjustment of the resources used by the application and either reduce in power consumption or improve in server utilization. The latter, in turn, allows avoiding resource over-provisioning and leads to better power efficiency in a DC.

IV. Microservices

Microservices is an innovative development style which allows to build applications composed by several small independent but interconnected modules [25][26]. Each module runs using its own processes, and communicate with other modules by means of a lightweight mechanism that typically consists of an HTTP-based REST API [27]. The result is an asynchronous, shared-nothing, highly scalable architecture that can be distributed over a possibly heterogeneous infrastructure. The ultimate goal is to avoid monstrous monolithic code that is poorly scalable and difficult to maintain and update.

Figure 1 depicts the general architecture of a microservice-based system. A management module runs on the hosting infrastructure (i.e., the set of hardware resources and the operating system). It is responsible for performing basic management operations on the microservices, such as the allocation and freeing of resources once they complete the assigned task. Each microservice is an entity isolated from the others. The communication and the data exchange among these entities rely on a public API (typically based on a HTTP REST protocol). This interface is also decoupled from the service core and its internal storage. Despite their isolation, microservices can still exploit basic services exposed by the host infrastructure.

The microservice architectural style is driven by concepts such as *low coupling*, that proposes to isolate functionalities in different modules, and *keep it simple*, that fosters the creation of small modules with a single responsibility. The need for a solution that can follow the technological trends which have shaken the IT landscape in the recent years led to their creation. In particular, the transformation of IT infrastructures caused by the advent of cloud computing gave great opportunities for developing software (e.g., scientific applications) that was able to take advantage of the elasticity and scalability of the underlying infrastructure [28][29]. However, the exploitation of these benefits requires more flexible software architectures, that enable applications to take advantage of the virtualized and distributed nature of the underlying infrastructure. Another important factor is the popularization of Linux containers, often referred to as lightweight virtualization. The main feature of this technology is the ability of multiple processes to share operating system resources, while still remaining isolated from each other. This characteristic allows the creation of virtual instances with less overhead with respect to traditional hypervisor-based VMs, meaning faster start-up time and smaller memory footprint. These two trends allow designing highly scalable applications composed by several redundant modules which are possibly distributed in a cloud infrastructure with heterogeneous nodes, thus ensuring performance, high-availability and scalability.

A. Benefits and drawbacks

One of the main benefit of the microservices-based architectures is the module *independence*. Each component represents a single function of the application and can be independently developed and deployed. Each development team works in autonomy and can choose the preferred or most appropriated platform and language. This means that microservices can be spread across heterogeneous infrastructures, composed of completely different platforms, run-time environments and hardware architectures. Further, each module is fully replaceable and upgradable, easing application maintenance.

Managing a large set of distributed components has its own issues. First of all, *service distribution* has a negative impact on the performance that is caused by the time for transferring data and the time to process requests. This latency is worsen by the *asynchronous* communication mechanisms used to exchange information. Another drawback derives from the use of distributed databases that should be synchronized instantaneously at the time changes are made. The result is the temporary *inconsistency* of information that must be taken into account by application developers.

B. Examples

Examples of applications that are organized as interconnected microservices include popular services like Twitter, eBay, Amazon, Netflix, and SoundCloud.

Amazon is one of the first big industry players to move to microservices, because it was afflicted by scalability problems since the e-commerce service was offered. Amazon began serving its e-commerce platform by using a monolithic application written in C++, but soon the database, the back-end and the front-end applications were restricted in their evolution due to the limitation of the shared architecture. These problems were addressed by isolating business logic and related data in independent applications. Over time, Amazon e-commerce

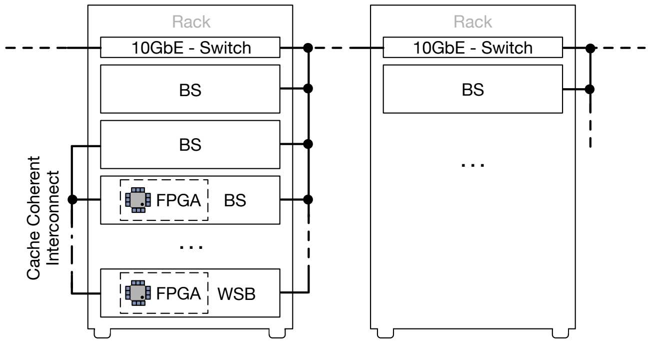

Fig. 2. The proposed hardware architecture model for OPERA. Racks are equipped with brawny servers (BS) and boxes of wimpy servers (WSB), while FPGA boards can be used to offload general purpose CPUs form specific computations.

platform has evolved in to hundreds of services and systems written in Java and Scala for aggregating information. The architecture become a fully-distributed, decentralized, service-based platform (PaaS) serving many different applications. Today Amazon web services, as well as sellers and customers use more than 100 different services to collect data and construct the final web page [30].

eBay was created in 1995 and in 2006 it managed more than 200 million users and more than one billion photos [31]. It started as a monolithic Perl application written as a single DLL, numbering more than 3.4 million lines of code. The first step towards the transformation into a microservices-based architecture was the code splitting in to separate functions written in Java. Today the Java core is still present in the system, but it is composed by a set of multi-lingual microservices.

SoundCloud is a popular social platform where users can create and share sounds [32]. Despite it being younger than other applications, it evolved over the time to become a microservices-based platform. The whole service is developed using 8 different programming languages, and the code management is centralized on a common GitHub repository. The service is split into several hundreds microservice instances (both stateless and stateful), which are managed through the Bazooka and Chef systems. The former is a deployment system built for the microservices that expose their interface via a HTTP REST API, and it is responsible for job scheduling, process supervision, service discovery setup, request handling and load balancing, as well as monitoring and logging. The latter is used to deploy all infrastructure microservices, including those managed by Bazooka.

V. OPERA Architecture

OPERA is a recently started research project, that is funded by EC under the Horizon 2020 framework. The main goal of the project is to exploit the potential use of heterogeneous computing platforms for achieving better power efficiency in DC infrastructures.

Figure 2 is a diagram showing a possible instance of the heterogeneous computing platform envisioned by OPERA. The left side of the figure shows the organization of a DC rack containing several different nodes. Compute nodes labeled as *brawny servers* (BS) represent traditional servers that would be commonly found in a today data center. They could be based on an architecture such as Intel's X86 or IBM's POWER. We remain agnostic with respect to the architecture (ISA) since we are mainly interested in applying general techniques for managing the software which runs on these servers. Since GNU Linux and many open source packages run on all the architectures in question, we consider them as equivalent, only accounting for their relative processing capabilities. What unifies BS (regardless of ISA) is a loose definition of capabilities [33]. These servers typically have multiple multicore processors, running in the 3-5 GHz clock range, and equipped with large amounts of DRAM (in the range of 4 to 16 GiB per core). Furthermore they may contain high-speed I/O devices such as SATA or SAS disks, and one or more 10 GbE network interface cards (NICs). In contrast to BS, the rack also contains boxes of *wimpy servers* (WSB). These servers are considerably less powerful than the brawny counterpart, and typically run in the 1-3 GHz clock range. However, they have the advantages of being more densely populated, drawing less power and creating less heat. There have been many discussions and studies about the possibility of replacing BS nodes with WSBs, but it appears that the drawbacks outweigh the benefits at this time. It is known that wimpy servers have better power efficiency than brawny servers (power consumption is relative to the square of the frequency), but not necessarily better energy proportionality.

A. Hardware heterogeneity

Generally, heterogeneity concerns the adoption of processing elements with different characteristics in terms of architecture, power consumption, and performance. Despite the use of a mix of brawny and wimpy servers (each with its own set of architectural features intended to maximize the performance and energy efficiency – e.g., the access to a vector execution unit), can be considered a form of heterogeneity, hardware differentiation can be wider. Despite that two processors support the same set of instructions (i.e., same ISA), their internal organization can be quite different. For instance, in-order and out-of-order processors can execute the same code but their performance are totally different, as well as their power consumption. Reconfigurable devices introduce a further level of differentiation. In this case a specific portion of code can be replaced by a dedicated logic circuits that performs better than its CPU-like counterpart. From this perspective, there are several possible combinations of brawny processors, wimpy processors, and accelerators that are worthy of investigation in the OPERA project.

We believe that future data centers will embrace hardware heterogeneity for two reasons. The first reason is that there is a benefit to using special-purpose hardware to solve a specific problem. General purpose hardware can be more convenient for programmers to solve a wide range of problems, but specific hardware is going to be more efficient at solving the particular problems that it is designed for. This includes to a lesser degree general purpose CPUs that were designed with different criteria, such as minimal power consumption or heat dissipation. The second reason is that heterogeneity is extremely difficult to avoid in a data center. As compute nodes fail and are replaced, the exact same hardware is likely no longer available as a replacement. The result is that there can be several versions or generations of compute nodes coexisting in a single data center. While it seems convenient from an IT perspective to mask these changes and treat all compute nodes as identical, this kind of policy can have negative results on application performance. For example, in parallel jobs, the running time is determined by the slowest node. If all nodes get an equal amount of work, the slower nodes will prolong the running time unnecessarily, while the faster nodes will wait idly for the slower nodes to complete. We believe that acknowledging the fact that different compute nodes have different strengths allows us to plan accordingly and take the differences into account when scheduling tasks.

Among the various forms of hardware acceleration, FPGAs have recently gained attention for their unique characteristics as an alternative to GPGPUs. Unlike conventional processors (CPUs or GPGPUs) where the software controls the way fixed hardware structures are activated in time, in a FPGA the hardware structure can be reorganized over the time (hardware reconfiguration), leading to the implementation of customized circuits. This feature can be used to efficiently execute portions of an application with a dedicated hardware circuit, and without the need for the software to direct control the circuit. A critical point of using such devices is the need of complex toolchains to transform a piece of code written in a high level language to a correctly working hardware circuit. However, custom circuits can be made available in the form of pre-synthesized IP blocks that can be instantiated also by an automated procedure every time a specific task needs to be accelerated, thus decoupling their implementation from their regular usage. Further, exploiting custom circuits directly translate into a more energy efficient execution of the application. In this context, the OPERA project is going to investigate the use of FPGAs for the creation of fast back-to-back communication between different compute nodes and for enabling cache coherency between CPUs of different architectures. Figure 2 shows an instance of this heterogeneous platform. The racks are equipped with both brawny and wimpy server boxes, while FPGA boards may be used for offloading certain calculations to custom circuits in order to achieve a better power/performance ratio when executing an application.

B. Microservices integration

We envisage the possibility of extensively expressing applications in the form of interconnected microservices, which are then automatically scheduled for execution on the most suitable computing elements.

However, the deployment of a set of interconnected tasks in a heterogeneous context is critical, since different processor architectures and accelerators must be used. To take advantage of the availability of diverse hardware elements, a mechanism to automatically assign tasks to the most suitable architecture must be put in place. To cope with this issue, we propose to adopt the microservices model in the following way: (i) the application is partitioned into independent tasks; (ii) each task is characterized in order to ease the mapping with the features of the available hardware; (iii) each task is wrapped by a thin module that exports the communication interface (e.g., a HTTP-based REST API); (iv) the communication among the microservices (e.g., the exchange of intermediate results of the calculation) takes place by means of the exported API. However, to make this process effective (see figure 3) a mechanism

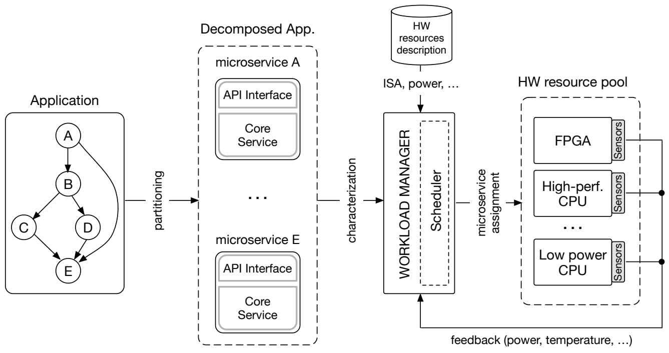

Fig. 3. Managing microservices running on a heterogeneous infrastructure: a workload manager assigns microservices to the most suitable hardware system, based on the power/energy feedback and hardware capabilities.

to describe the microservices in terms of their relationships with others and interactions with various layers of the cloud infrastructure is needed. Furthermore, feedback information coming from the servers executing the microservices instances should be collected, so that a correct allocation and scheduling policy (i.e., decide where a specific microservice should be run in order to maximize performance and energy efficiency) can be applied.

The first issue can be solved by describing different modules composing the microservices through an application descriptor, which allows the abstraction of different components of cloud applications, as well as describes their relationships and interconnections. This artifact enables the portability of cloud applications and services across different platforms. Among the various standards, OASIS TOSCA [34] is one of the most complete, and it has recently been extended to support containers (specifically it allows to describe an application in terms of interconnections among various modules, energy consumption and hardware requirements). We think it could be the most suitable candidate for serving this purpose.

Addressing the second issue requires the ability of scheduling microservices depending their specific requirements (e.g., specific hardware, minimum performance requirement, etc.) and policies for maximizing energy efficiency of the whole cloud infrastructure. The design and implementation of such a system, here called *workload manager*, is part of the research plan of the OPERA project. This component is in charge of scheduling and assigning microservices to the different hardware components, by satisfying the following criteria:

1) Must be able to monitor energy/power efficiency of each node;
2) Must be able to quickly react to the changes in workload demand;
3) Must be aware of each node's capabilities and specific features.

The first criteria can be easily met, since modern hardware contains embedded sensors to allow monitoring of several operational parameters (e.g., core temperature, speed of fans, etc.). Furthermore, several tools are available to monitor the workload of the nodes. The second criteria can be satisfied by leveraging the capability of the underlying infrastructure to scale up and down the allocated resources. Finally, the third criteria is simply satisfied by adopting a standardized way of describing application requirements and capabilities of each nodes.

C. Legacy applications

Despite the clear benefits that microservices can provide, we must also acknowledge the existence of many applications that are not written following the microservice principles. These applications include those that are monolithic but written for the cloud (e.g., Zimbra), and others that were written before the cloud paradigm became popular (e.g., DB2). While in the former case, applications can be ported to our development model (eventually a more coarse level of splitting the application in independent tasks can be used), in the latter case applications (we call them legacy applications) cannot be easily and directly adapted and thus they will not be able to take advantage of modern features like auto-scaling. Since there are still many of such applications, we should provide a strategy to manage them. Such a strategy is part of the research plan of the OPERA project.

VI. Conclusion

The appeal of using cloud computing is the ability for an application to automatically scale according to the changes of the workload demand. However, this is made possible at the cost of a large energy inefficiency of the underlying infrastructure. To overcome this inefficiency, heterogeneous hardware has started to appear as standard equipment of data center infrastructures. Limitations in the full exploitation of such heterogeneity comes from the difficulties of mapping cloud applications with the different hardware components. This paper proposes a way to overcome these limitations by adopting a highly scalable model for developing applications. Before, describing the model, the characteristics of cloud workloads along with their opportunity of gaining energy efficiency are analyzed. The proposed model achieves such scalability by partitioning the application into independent tasks which remain isolated each other, and that communicate through a standard interface. We introduced this model in the context of the OPERA project, by exposing desired features of the workload manager. We conclude the paper by highlighting the need of taking care of legacy applications that are by nature not portable to the proposed model and cannot take advantage of running on a heterogeneous environment in terms of performance, scaling and energy efficiency.

Acknowledgment

This work is part of the OPERA project, which has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 688386

References

[1] Jonathan Koomey. Growth in data center electricity use 2005 to 2010. *A report by Analytical Press, completed at the request of The New York Times*, page 9, 2011.

[2] Yongmei Xu, Zeqi Gao, and Yuhui Deng. Analyzing the cooling behavior of hot and cold aisle containment in data centers. In *Emerging Intelligent Data and Web Technologies (EIDWT), 2013 Fourth International Conference on*, pages 685–689, Sept 2013.

[3] Jing Mei and Kenli Li. Energy-aware scheduling algorithm with duplication on heterogeneous computing systems. In *Grid Computing (GRID), 2012 ACM/IEEE 13th International Conference on*, pages 122–129, Sept 2012.

[4] J Whitney and P Delforge. Data center efficiency assessment–scaling up energy efficiency across the data center industry: Evaluating key drivers and barriers. *NRDC and Anthesis, Rep. IP*, pages 13–14, 2014.

[5] J.R. Stanley, K. Brill, and J. Koomey. Four metrics define data center greenness enabling users to quantify energy effi ciency for profit initiatives.

[6] Luiz Andr Barroso, Jimmy Clidaras, and Urs Hlzle. The datacenter as a computer - an introduction to the design of warehouse-scale machines second edition. 2013.

[7] S.P. Crago and J.P. Walters. Heterogeneous cloud computing: The way forward. *Computer*, 48(1):59–61, Jan 2015.

[8] Jose Nunez-Yanez. Energy efficient reconfigurable computing with adaptive voltage and logic scaling. *SIGARCH Comput. Archit. News*, 42(4):87–92, December 2014.

[9] G. Durelli, M. Coppola, K. Djafarian, G. Kornaros, A. Miele, M. Paolino, Oliver Pell, Christian Plessl, M. D. Santambrogio, and C. Bolchini. *Reconfigurable Computing: Architectures, Tools, and Applications: 10th International Symposium, ARC 2014, Vilamoura, Portugal, April 14-16, 2014. Proceedings*, chapter SAVE: Towards Efficient Resource Management in Heterogeneous System Architectures, pages 337–344. Springer International Publishing, Cham, 2014.

[10] Christina Delimitrou and Christos Kozyrakis. Paragon: Qos-aware scheduling for heterogeneous datacenters. *ACM SIGARCH Computer Architecture News*, 41(1):77–88, 2013.

[11] David G Andersen, Jason Franklin, Michael Kaminsky, Amar Phanishayee, Lawrence Tan, and Vijay Vasudevan. Fawn: A fast array of wimpy nodes. In *Proceedings of the ACM SIGOPS 22nd symposium on Operating systems principles*, pages 1–14. ACM, 2009.

[12] Vinicius Petrucci, Michael A Laurenzano, John Doherty, Yunqi Zhang, Daniel Mosse, Jason Mars, and Lingjia Tang. Octopus-man: Qos-driven task management for heterogeneous multicores in warehouse-scale computers. In *High Performance Computer Architecture (HPCA), 2015 IEEE 21st International Symposium on*, pages 246–258. IEEE, 2015.

[13] Jeffrey S. Chase, Darrell C. Anderson, Prachi N. Thakar, Amin M. Vahdat, and Ronald P. Doyle. Managing energy and server resources in hosting centers. *SIGOPS Oper. Syst. Rev.*, 35(5):103–116, October 2001.

[14] Ripal Nathuji and Karsten Schwan. Virtualpower: coordinated power management in virtualized enterprise systems. In *ACM SIGOPS Operating Systems Review*, volume 41, pages 265–278. ACM, 2007.

[15] David Meisner, Brian T. Gold, and Thomas F. Wenisch. Powernap: Eliminating server idle power. *SIGARCH Comput. Archit. News*, 37(1):205–216, March 2009.

[16] Susanne Albers and Antonios Antoniadis. Race to idle: New algorithms for speed scaling with a sleep state. *ACM Trans. Algorithms*, 10(2):9:1–9:31, February 2014.

[17] Race-to-idle. 2012.

[18] Anton Beloglazov, Jemal Abawajy, and Rajkumar Buyya. Energy-aware resource allocation heuristics for efficient management of data centers for cloud computing. *Future generation computer systems*, 28(5):755–768, 2012.

[19] David Meisner, Christopher M Sadler, Luiz André Barroso, Wolf-Dietrich Weber, and Thomas F Wenisch. Power management of online data-intensive services. In *Computer Architecture (ISCA), 2011 38th Annual International Symposium on*, pages 319–330. IEEE, 2011.

[20] David Lo, Liqun Cheng, Rama Govindaraju, Luiz André Barroso, and Christos Kozyrakis. Towards energy proportionality for large-scale latency-critical workloads. In *Proceeding of the 41st Annual International Symposium on Computer Architecture*, ISCA '14, pages 301–312, Piscataway, NJ, USA, 2014. IEEE Press.

[21] Jason Mars, Lingjia Tang, Robert Hundt, Kevin Skadron, and Mary Lou Soffa. Bubble-up: Increasing utilization in modern warehouse scale computers via sensible co-locations. In *Proceedings of the 44th Annual IEEE/ACM International Symposium on Microarchitecture*, MICRO-44, pages 248–259, New York, NY, USA, 2011. ACM.

[22] Hailong Yang, Alex Breslow, Jason Mars, and Lingjia Tang. Bubble-flux: Precise online qos management for increased utilization in warehouse scale computers. In *Proceedings of the 40th Annual International Symposium on Computer Architecture*, ISCA '13, pages 607–618, New York, NY, USA, 2013. ACM.

[23] Marco Guazzone. Mining the workload of real grid computing systems. *CoRR*, abs/1412.2673, 2014.

[24] Arun Babu Nagarajan, Frank Mueller, Christian Engelmann, and Stephen L. Scott. Proactive fault tolerance for hpc with xen virtualization. In *Proceedings of the 21st Annual International Conference on Supercomputing*, ICS '07, pages 23–32, New York, NY, USA, 2007. ACM.

[25] Sam Newman. Building microservices : designing fine-grained systems, 2015.

[26] Johannes Thones. Microservices. *Software, IEEE*, 32(1):116–116, Jan 2015.

[27] Representational state transfer. 2000.

[28] Guilherme Galante, Luis Carlos Erpen De Bona, Antonio Roberto Mury, Bruno Schulze, and Rodrigo Rosa Righi. An analysis of public clouds elasticity in the execution of scientific applications: a survey. *Journal of Grid Computing*, pages 1–24, 2016.

[29] Michael Armbrust, Armando Fox, Rean Griffith, Anthony D. Joseph, Randy Katz, Andy Konwinski, Gunho Lee, David Patterson, Ariel Rabkin, Ion Stoica, and Matei Zaharia. A view of cloud computing. *Commun. ACM*, 53(4):50–58, April 2010.

[30] WERNER VOGELS JIM GRAY. A conversation with werner vogels, learning from the amazon technology platform. 2006.

[31] Dan Pritchett Rnady Shoup. The ebay architecture, striking a balance between site stability, feature velocity, performance and cost. 2006.

[32] Johan Uhle. On dependability modeling in a deployed microservice architecture. Master's thesis, University of Potsdam, Germany, 2014.

[33] Urs Hlzle. Brawny cores still beat wimpy cores, most of the time. 30, 2010.

[34] Oasis topology and orchestration specification for cloud applications (tosca) tc. 2015.