The Green Computing Continuum: the OPERA Perspective

A. Scionti, O. Terzo, P. Ruiu, G. Giordanengo, S. Ciccia, G. Urlini, J. Nider, M. Rapoport, C. Petrie, R. Chamberlain, G. Renaud, D. Tsafrir, I. Yaniv, and D. Harryvan

Abstract Cloud computing is an emerging paradigm in which users access to a shared pool of computing resources dynamically allocated (i.e., ubiquitous computing service), depending on their specific needs. Such paradigm exploits the infrastructural capabilities of modern data centers to provide computational power and storage space required to satisfy modern application demands. The seamless integration of Cyber-Physical Systems (CPS) and Cloud infrastructures allows the effective processing of the huge amount of data collected by smart embedded systems, towards the creation of new services for the end users. However, trying to continuously increase data center capabilities comes at the cost of an increased energy consumption. The OPERA project aims at bringing innovative solutions to increase the energy efficiency of Cloud infrastructures, by leveraging on modular, high-density, heterogeneous and low power computing systems, spanning data center servers and remote CPS. The effectiveness of the proposed solutions is demonstrated with key scenarios: a road traffic monitoring application, the deployment of a virtual desktop infrastructure, and the deployment of a compact data center on a truck.

A. Scionti · O. Terzo · P. Ruiu · G. Giordanengo
ISMB, Torino, IT. e-mail: {scionti,terzo,ruiu,giordanengo}@ismb.it

S. Ciccia
Politecnico di Torino, Torino, IT. e-mail: simone.ciccia@polito.it

G. Urlini
STMicroelectronics, Milano, IT. e-mail: giulio.urlini@st.com

J. Nider · M. Rapoport
IBM Research - Haifa, IL. e-mail: {joeln,rapoport}@il.ibm.com

C. Petrie · R. Chamberlain
Nallatech Ltd, Glasgow, UK. e-mail: {c.petrie,r.chamberlain}@nallatech.com

G. Renaud
HPE, Grenoble, FR. e-mail: gallig.renaud@hpe.com

D. Tsafrir · I. Yaniv
Technion – Israel Institute of Technology, IL. e-mail: {dan,idanyani}@cs.technion.ac.il

D. Harryvan
Certios, NL. e-mail: dirk.harryvan@certios.nl

1 Introduction

More powerful and smart computing systems are made possible by the continuous advancements in silicon technology. Embedded systems evolved into modern Cyber-Physical Systems (CPS): smart connected system that are enough powerful to enable (near) real-time interactions with the surrounding environment. For this reason, CPS are at the basis of implementing new services, albeit they generate an enormous amount of data, thanks to their capability of sensing/acting on the environment where they are deployed. Cloud computing, or simply Cloud, is the set of hardware and software technologies used to process and analyse such amount of data, so that it becomes possible to respond to the societal and industrial needs through innovative services. Also, Cloud technologies enables CPS to retrieve useful information to react to the changes in the environment where they operate. However, such welcome capabilities are today counterbalanced by the high power consumption and energy inefficiencies of the processing technologies that traditionally power data center (DC) servers. Generally, DCs have thousands of running servers arranged in multiple racks. Maintaining these infrastructures has high costs, and Cloud providers pay 40% to 50% of their total expenses as power bills [2]. The critical point of these large scale infrastructure is represented by their large inefficiency: the average server utilisation remains between 10% to 50% in most of the DCs [3, 4] and further, idle servers can consume up to 65% of their peak power [3, 5, 6]. New architectural solutions are thus required to provide more responsiveness, scalability, and energy efficiency.

One way of ameliorating the situation is to use virtualization techniques [7], which increase server utilisation by replicating many times the behaviour of a physical machine, by means of the abstraction of the main hardware features. Thanks to virtualization, servers with different resources can dynamically host different "virtual machines" (VMs). Hypervisors, i.e., software devoted to control VMs during their lifetime, play a key role in scheduling the allocation of VMs to the physical servers. Also, hypervisors consume computing resources for achieving their goals, which in turn translates into power consumption. In addition, traditional virtualization systems put large pressure on the servers' memory subsystem, thus further contributing to increase power consumption, and limiting the capability of the hypervisors to pack a large number of VMs on the servers. Cloud paradigm allows acquiring and releasing resources dynamically over time, making the problem of allocating computing and storage resources even more complex. The stochastic nature of Cloud workloads, along with blind resource allocation policies (i.e., resources are usually provisioned to their peak usage) employed by most DCs, lead to poor resource utilisation [8, 9]. Thus, there is a demand for more efficient resource allocation mechanisms, capable of exploiting the large architectural heterogeneity available in modern DCs, as well as capable of leveraging on less memory hungry virtualization systems. Furthermore, such allocation strategy should be helpful in reducing the workload imbalance and also to optimise resource utilisation.

Tackling these challenges, the OPERA project [1] aims at investigating on the integration of high-performance, low power processing elements within highly mod-

ular and scalable architectures. OPERA realises that heterogeneity is a key element for achieving new levels of energy efficiency and computational capabilities. To this end, solutions delivered in the project widely leverage on reconfigurable devices (field programmable gate arrays – FPGAs) and specialised hardware modules to maximise performance and reduce power consumption. Also, improving the efficiency of the virtualization layer represents one of the main objectives of the OPERA project. Improving the way memory is consumed by (virtualized) Cloud applications, adopting more lightweight technologies (e.g., Linux containerisation), and better supporting allocation of heterogeneous computing resources, OPERA aims at greatly reducing the overall energy consumption of next generation Cloud infrastructures, as well as of remotely connected CPS. Specifically, the following objectives have been identified:

- Exploiting heterogeneity through the adoption of different multicore processor architectures (i.e., ARM, X86_64, and POWER), along with specialised accelerators based on reconfigurable devices (FPGAs);
- Automatically splitting workloads in to a set of independent software modules which can be executed as "microservices" on specific target devices, in order to improve the overall energy-efficiency of the Cloud infrastructure;
- Leveraging on direct optical links and the Coherent Accelerator Processor Interface (CAPI) to increase scalability and maximise the performance of the Cloud infrastructure;
- Designing a scalable, modular, and standardised server enclosure (also referred to as "Data Center-in-a-Box") supporting tens to hundreds of processing cores (even with different architectures) in a very compact form-factor;
- Defining and integrating appropriate metrics with a more holistic approach to continuously monitor the efficiency of the Cloud computing platform;
- Exploiting ultra-low power multi-/manycore processors with a dedicated energy-aware instruction set architecture (ISA) to improve processing CPS capabilities in the context of computer vision applications;
- Exploiting reconfiguration and adaptability of the radio communication subsystem to enhance the CPS efficiency;
- Introducing offloading services (possibly accelerated through FPGA boards) to enable remote CPS accessing larger computing capabilities for critical tasks.

The project, spanning over 3 years, is coordinated by STMicroelectronics, along with the support of Istituto Superiore Mario Boella – ISMB (technical coordinator). The consortium involves also industrial partners, as well as public bodies and academic institutions (HPE, Nallatech, IBM, Teseo–Clemessy, Certios, CSI Piemonte, Le Département de Isère, Neavia technologies, Technion) to successfully achieve the above mentioned objectives.

In this chapter, we discuss the set of technologies, architectural elements, and their integration in the OPERA solution, which are the results of the research activity carried out in the project. Specifically, the design, and integration of a high-density, highly scalable, modular, server equipped with acceleration boards is presented, as well as the capability of integrating, at the DC level, HPC-oriented machines

equipped with POWER processors and FPGA accelerators. The integration of an energy-aware Cloud orchestration software is analysed, along with the analysis of the application impact on the memory subsystem, which is at the basis of a more energy-aware optimisation of the running applications. Finally, the extension of the Cloud infrastructure to include new smart and energy efficient CPS is presented: thanks to network connections, a mechanism for offloading some of the computation on the Cloud back-end is discussed as well.

The rest of the chapter is organised as follows. Section 2 introduces state-of-the-art related works; Section 3 gives an overview of Cloud paradigm evolution. In Section 4 and Section 5, we detail OPERA infrastructure resources, spanning from DC servers to remote CPS. Next, in Section 6 we describes the three real-life application scenarios where OPERA solutions is successfully applied. Finally, we conclude the presented chapter in Section 7.

2 Related works

The OPERA project exploits many different technologies with the ambition of integrating them into a more (energy) efficient platform, serving as the basis for creating the next generation Cloud infrastructures.

Improving energy efficiency in Cloud domain has seen several approaches being proposed in recent years [10]. Most of the proposed approaches aim at improving resource allocation strategies, targeting a more efficient use of the computing and storage resource, which in turn translates in reducing energy costs. Various algorithms have been used to implement smarter resource allocation strategies, including: best fit [11], first fit [12], constraint satisfaction problem (CSP) [13], control theory [14], game theory [15], genetic algorithms (GA) and their variants [16], queuing model [17], and also various other heuristics [18, 19]. It is interesting to note that greedy algorithms (e.g., best fit and first fit) are employed in commercial products, such as the distributed resource scheduler (DRS) tool from VMware [20]). As an emerging architectural style in developing Cloud applications, splitting them in to a set of independent microservices, it becomes possible to control their allocation on the Cloud resources with a fine-grain resolution with regards to traditional monolithic services, as well as it becomes possible to predict their incoming.

Resource allocation can be conveniently formalised as a Bin Packing Problem (BPP), which is a NP-hard optimisation problem. Generally, solving large instances of BPP requires the adoption of heuristics. Among the various, evolutionary-based algorithms, such as Particle Swarm Optimization (PSO), provide very promising results. Zhan et al. [31] proposed a hybrid PSO algorithm by combining the standard PSO technique with the simulated annealing (SA) to improve the iterations of the PSO algorithm. In [33], authors introduced a mutation mechanism and a self-adapting inertia weight method to the standard PSO to improve convergence speed and efficiency. Zhang et al. [32] describe a hierarchical PSO-based applica-

tion scheduling algorithm for Cloud, to minimise both the load imbalance and the inter-network communication cost.

Time series based prediction has been also used as a mean for foreseeing Cloud workloads in advance, so that it is possible to reserve and optimally allocate DC resources. A comprehensive set of literature is available in this context. A simplified method for modelling univariate time series is given by autoregressive (AR) model. Combined with the moving average (MA), it leads to the well-known ARMA model. In [21], authors proposed a ARMA-based framework for characterising and forecasting the daily access patterns of YouTube videos. Tirado et al. [22] present a workload forecasting AR-based model that captures trends and seasonal patterns. Prediction was used to improve the server utilisation and load imbalance by replicating and consolidating resources. Other works proposed AR-derived models to dynamically optimise the use of server resources and improve energy efficiency [24]. Similarly, the autoregressive integrated moving average (ARIMA) and autoregressive fractionally integrated moving average (ARFIMA) can be used to capture non-stationary behaviours in the time series (i.e., it performs better in describing mean, variance, autocorrelation of time series, which in turn change over time), and to extract statistical self-similar (or long memory) patterns embedded in the data sets. To this end, several research works have been presented [25, 26]. On the other hand, it is also a common practice for the Cloud providers to automatically provisioning virtual machines (VMs) via a dynamic scaling algorithm, without using any predictive method [27, 28].

Heterogeneity (including GPGPUs, FPGAs, Intel XeonPhi, many-cores) have been popularised in HPC domain to accelerate computations and to lower power consumption costs of computing infrastructures. Thanks to different architectural features of the processing elements, heterogeneous solutions allow to better adapt to the workload characteristics. However, they still are not common in the Cloud computing domain [34], where architectural homogeneity helps to reduce the overhead and costs of managing large infrastructures. Although, Cloud providers started offering instances running on powerful GPGPUs (e.g., Amazon AWS G2 and P2 instances, Microsoft Azure N-type instances), the access to FPGA-based images is still complex and very limited, since the difficulties in programming such devices. Some steps towards easing the programming of FPGAs has been done with language extensions and frameworks supporting the translation of code written with popular high-level languages (C/C++) into synthesisable ones (VHDL/Verilog). Examples of such compilation frameworks are represented by LegUp [41], ROCCC [42], and OpenCL [43]. However, the lack of precise control of the placement and routing of FPGA resources, generally still demands for manual optimisation of the final design. Despite many challenges to address, some works tried to reverse the situation, explicitly targeting Cloud workloads [35, 36, 37]. However, such works only provide a glimpse of what future evolution of programming languages and compilers will enable (in fact, most of this works strongly rely on an hand-made design optimisation phase).

Cloud-connected smart sensors (also referred to as Cyber-Physical Systems) are often demanded for processing intensive tasks to avoid large data transfer to the

Cloud back-end. To this end, CPS are equipped with high-performance processors, which run in an ultra-low power envelop. Examples of such design style are PULPino [38], an ultra-low power architecture targeting IoT domain, and the ReISC core architecture [39]. To achieve high-performance within an ultra-low power envelope, CPS processors exploits a mix of energy-optimised technologies, including an energy-aware instruction set architecture (ISA). Despite manufacturing technology and micro-architectural improvements, many application scenarios require computational capabilities (e.g., real-time elaboration of high-resolution streaming videos) that are far from that offered by an CPS. In that case, offloading computational intensive task to a remote Cloud back-end becomes the only feasible solution. RAPID EU-funded project [44] aims at accelerating the capabilities of low power embedded devices by taking advantage of high-performance accelerators and high-bandwidth networks. Specifically, compute or storage intensive tasks can be seamlessly offloaded from the low-power devices to more powerful heterogeneous accelerators. For instance, such mechanism has been successfully used to enable Android devices with no GPU support to run GPU-aware kernels, by migrating their execution on remote servers equipped with high-end GPGPUs [45]. Communication capabilities (both wired and wireless) are at the basis of the implementation of effective offloading services, and remain largely exploited in CPS deployed in unmanned contexts. Adapting the communication subsystem to the physical channel characteristics [40] greatly help the deployment of CPS also in rural areas, as well as greatly contributes to save energy on the battery.

3 The green computing continuum

Cloud computing (or simply Cloud) provides a way to access computing resources without hosting them on premise. The level of abstraction at which computing resources are accessed (i.e., the way Cloud users interact with the infrastructure) determines the adopted Cloud service model. Three main service models exist: Software-as-a-Service (SaaS), Platform-as-a-Service (PaaS), and Infrastructure-as-a-Service (IaaS). OPERA aims at improving the way services and resources are managed within a DC, thus it mostly considers the IaaS model. At this level of abstraction (i.e., users may deal with the low level infrastructure, by requesting virtual machines and managing their interconnections) resource allocation problem emerges as one of the main challenges to address. In addition, recently we witnessed to the broadening of Cloud infrastructures, beyond their traditional concept. With the fast growth of the number of small embedded devices connected to Internet [46], Cloud providers need to support them anywhere at any time. The infrastructural Cloud support helps to integrate Cloudlets, Fog computing and the “Cloudification” of the Internet of Things (IoT) within traditional infrastructures (see Figure 1). Supporting such growing set of connected devices exacerbate the need for a more efficient way of managing and controlling resources within large-scale computing systems.

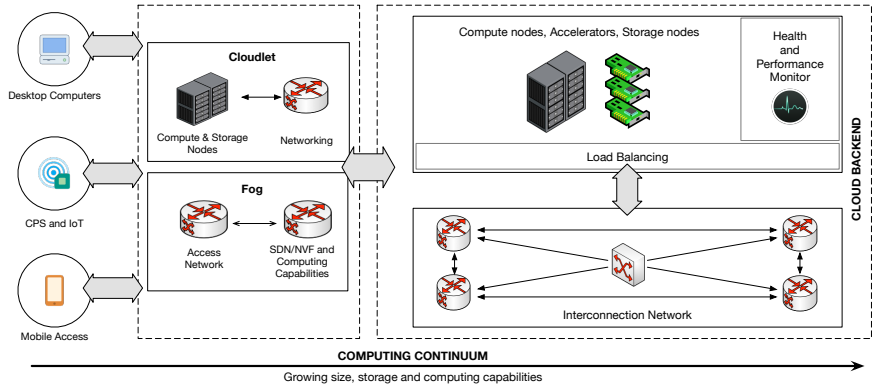

Fig. 1 A visual representation of the computing continuum: from left (smart embedded devices) to right (public cloud data centers), computing and storage capabilities grow. Along with compute and storage, energy consumption grows as well.

Nowadays, mobile computing has gained a momentum thanks to the progress in lowering power consumption of embedded hardware platforms. The higher the computing capabilities are, the more complex the applications running become. Thus, although the improvements in the capabilities of such platforms, Cloud becomes a choice of worth in supporting the remote execution of computing-intensive jobs every time the mobile device is not able to provide enough computational resources. To this purpose, Cloud infrastructures began to reduce the latency for processing jobs and to support (near) real time services. The adopted solutions, termed as Cloudlets [47], are trusted, capable systems co-located with the point-of-presence, and equipped with a fixed number of powerful servers. Using Cloudlets, users can instantiate, on-demand, custom VMs on a cluster of commodity machines, without the need for accessing traditional Cloud services. Fog computing is another emerging paradigm [48]. Here, the idea is to extend the Cloud infrastructure perimeter, by transferring and processing jobs within the network, aiming at reducing the access latency and improving QoS. Fog computing exploits the "virtualization" of the network equipment: router and switch functionalities are melded with more general purpose processing capabilities exposed through specific services [49]. The result of such transformation is a set of new technologies, such as software-defined networking (SDN) and network function virtualization (NFV) [50], which are emerging as a front runner to increase network scalability and to optimise infrastructural costs. SDN and NVF are also two examples of Cloud infrastructure elements that can strongly benefits from more efficient, high-density, computing machines (e.g., FPGA devices represents an optimal substrate for accelerating latency-sensitive network operations). Specifically, SDN provides a centralised, programmable network framework that can dynamically provision a virtualized server and storage infrastructure in a DC. SDN consists of SDN controllers (for a centralised view of the overall network), southbound APIs (for communicating with the switches and routers) and northbound APIs (to communicate with the applications and the busi-

ness logic). NFV helps to run network functions, such as network address translation (NAT) and firewall services as a piece of software. Another extension of the traditional Cloud infrastructures is represented by the Mobile Edge Computing (MEC) [51] paradigm. Here, the aim is to provide a dynamic content optimisation and also to improve the quality of user experience, by leveraging radio access networks (RANs) to reduce latency and increasing bandwidth.

The lowering cost of manufacturing technology led to integrate large computing capabilities, sensors and actuators within the same systems, unleashing the full potential of embedded systems as "cyber-physical systems" (CPS). The large availability of smart sensors/actuators and the corresponding emerging of the Internet-of-Things (IoT) paradigm [52], makes Cloud infrastructures necessary to store and process the enormous amount of collected data. Cloud-IoT applications are quite different compared to the traditional Cloud applications and services (due to a diverse set of network protocols, and the integration of specialised hardware and software solutions). From this standpoint, Cloud-based IoT platforms are a conglomeration of APIs and machine-to-machine communication protocols, such as REST, WebSockets, and IoT-specific ones (e.g., message queuing telemetry transport – MQTT, constrained application protocol – CoAP).

3.1 Energy efficiency perspective

Energy efficiency began an important topic for continuing to sustain the adoption of Cloud and IT technologies. Europe, as other countries, defined the objectives that must be achieved to make Cloud and IT technologies "green". To drive energy efficiency in the EU member states, targets have been detailed in the energy efficiency directive, EED [53]. This energy efficiency directive establishes a set of binding measures to help the EU reaching its energy efficiency target by 2020. Under this directive, all EU countries are required to use energy more efficiently at all stages of the energy chain, from its production to its final consumption. Interestingly, energy intensity in EU industry decreased by almost 19% between 2001 and 2011, while the increased use of IT and Cloud services by both individuals as well as organisations has resulted in a marked increase in datacenter energy use [54]. It is important to note that to decrease final energy use, either total production (output) must decrease and/or efficiency increases must outpace the increase in production.

Although the term "energy efficiency" (E_e) is used often in everyday life, the term warrants careful definition. In mathematical form all energy efficiency metrics are expressed as:

$$E_e = \frac{F_u}{E_{out}} \tag{1}$$

where F_u defines the functional unit (i.e., work done), and E_{out} represents the energy used to produce the output results. For the sake of correctly modelling energy efficiency, there are however many possibilities for describing *i*) energy consump-

tion, *ii*) system boundaries, and *iii*) functional units (this is mostly represented by the type of workload executed) and test conditions.

Looking at the energy consumption of DC equipment, to get an estimate, one must turn to one or more publically available data sources. These data sources are collected as LCI and LCA databases. LCI databases provide Life Cycle Inventory data-sets, while LCA databases include in addition Life Cycle Impact Assessments methods. In environmental impact studies, energy is most often expressed as primary energy (PE). Aside from the grid conversion, LCA studies add other external factors, such as datacenter cooling, into the PE calculation.

System boundaries are necessary to define in order to correctly calculate the energy efficiency of the system under evaluation. It implies to consider only the elements that actually influence the energy behaviour of the system. Further, workload composition becomes crucial to analyse the efficiency of DC equipment. To this standpoint, three main hierarchical levels can be defined. *System boundary*: this level incorporates all the equipment used in delivery of a service, including end-user equipment and transport networks; *Equipment boundary*: this level is a limitation of the system level, focused on a single machine or a tightly knit cluster of machines delivering certain functionalities (both the user and networking are excluded in this view); *Component boundary*: this level is a limitation of the equipment level focused on an element or component inside a machine or the cluster that performs a distinct function (a component view is not limited to a hardware component but can also be a software component).

Workload greatly influence the behaviour of the hardware units, which in turn determines their consumed energy. To demonstrate the effect of workload, it is useful to examine the results coming from the SPECpower benchmark. Interestingly, low server utilisation is common, even in the case of high load conditions. Similar outcomes remain valid for the Amazon EC2 service [55] where large fluctuations are present, but the long term average utilisation is in the range of 10% to 15%. In this range, even with an aggressive power management, servers do not perform efficiently. Not only workload volume influences the efficiency of computations, but also its composition. Different software components (possibly using different technologies, such as Java virtual machines, compiled languages, etc.) perform differently on a given machine, depending on the other software concurrently running. The influence of workload composition is harder to quantify than that of the workload volume. The issue being that what is measurable in a server is the power drawn along with the application output. Varying the workload composition modifies the application output, making comparison of results impossible. However, such differentiation can be turned into an advantage if heterogeneous hardware elements are used. In this case, it becomes possible to search for an advantage in bindings different workload components on different hardware elements, for better performance and energy reduction.

OPERA aims at providing a substantial improvement of the energy efficiency of a DC, by considering workloads made of software components used to deliver end-user services (e.g., access to remote virtual desktops, SaaS-based applications, offloading services, etc.). The influence of the workload volume and composition on

the computing elements is also investigated to better tune the architectural design, as well as the adopted technologies. The boundary is represented by the set of machines that perform computations, including also networking equipment and remote CPS (Cloud end-nodes). Thanks to a wide use of heterogeneous platforms, OPERA aims at greatly improving overall energy efficiency.

4 Heterogeneous data centers

Cloud computing paradigm is based on the availability of a large set of compute, storage and networking resources, which are in general heterogeneous in nature.

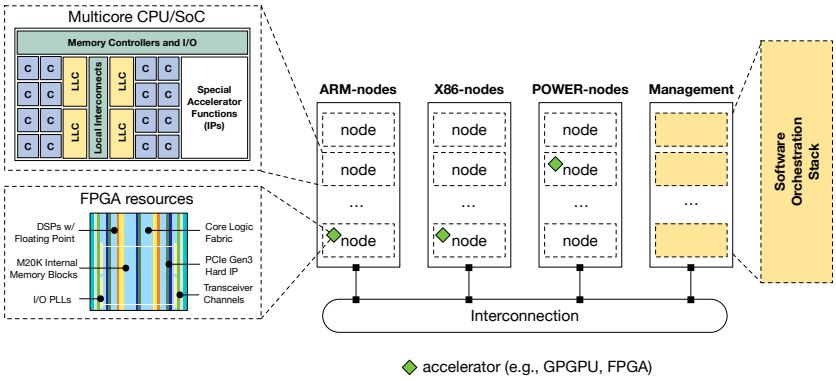

Fig. 2 Conceptual representation of a modern heterogeneous data center. Nodes have different instruction sets and micro-architectures, and can be accelerated through GPGPUs or FPGAs. Internal organisation of multicores and FPGAs is provided (left side).

Unlike in the past, the growing demand for more energy-aware systems, is leading data centers to rapidly embrace of different processor architectures and dedicated accelerators. The former are well represented by ARM-based systems, while the latter are represented by GPGPUs, with a smaller presence of FPGA devices. In OPERA, this practice is further extended by introducing also POWER-based systems. This is a platform originally intended for high-performance computing machines; however, with the emerging of Cloud services supporting HPC-oriented and scientific applications, the availability of dedicated computing nodes becomes more valuable. It is worth to highlight that heterogeneity in data centers extends in several dimensions, not only across architectures: machines configured in the same way and using the same CPU architecture can be still different each other since CPUs families change the features over generations (e.g., the most recent Intel Xeon processors support the AVX-512 instruction set extension which was not available on previous models), as well as the supported clock frequencies.

Figure 2 depicts the conceptual representation of such modern "heterogeneous" data center, highlighting the presence of software stack devoted to management. Looking at the internal organisation of modern chips, one can see the presence of multiple computing cores, along with dedicated interconnections and accelerating IPs (i.e., circuits providing specific functionalities in hardware). One of the main task in governing such kind of infrastructures regards the allocation of resources for the different applications. In order to be "efficient", such allocation should consider several factors to get the optimal allocation decision. To this purpose, in OPERA, power consumption of each platform and the relative load are considered good estimators of the relative energy consumption.

4.1 Accelerated servers

Figure 3 depicts the organisation of the heterogeneous DC as envisioned by OPERA. Hardware differentiation provides the proper computational power required by complex tasks running on top of diverse frameworks (e.g., Apache Hadoop, Apache Spark, MPI, etc.).

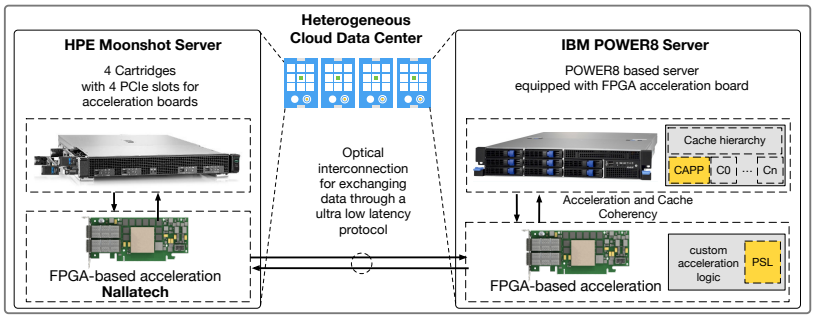

Fig. 3 OPERA envisioned heterogeneous Cloud data center infrastructure: high density, high performance and low power server nodes are linked through optical interconnect (via FPGA cards) with POWER-based nodes. FPGA provides also acceleration resources for computational heavy tasks.

To this end, OPERA integrates low power and high-performance processor architectures (ARM, X86_64, and POWER) into highly dense and interconnected server modules. Developed enclosures (HPE Moonshot) makes it possible to integrate hundreds of different processing elements by exploiting a microserver design: a single cartridge contains the specific processing element (i.e., CPU or DSP), the main memory and the storage. Each cartridge can be coupled with a dedicated accelerator (i.e., GPGPU, FPGA-based board, manycore device). Besides Cloud-specific applications, other algorithms may largely benefit from FPGA acceleration, such as network communication functions and protocols.

Effective usage of hardware components depends on the easiness in accessing their functionalities at the software level. To this end, in OPERA, FPGA accelerators are wrapped by an optimized Board Support Package (BSP) that deals with the low-level details of the FPGA architecture and peripherals (e.g., PCIe, CAPI, SerDes I/O, SDRAM memory, etc.), and that fits into the high-level OpenCL toolflow [57]. This allows the application (or a portion) to be represented as highly portable kernels, that Cloud orchestrator can decide at the last minute eventually to off-load from software running on the host processor to silicon gates. Furthermore, such designed accelerators furnish the DC servers with PCIe and Coherent Accelerator Processor Interface (CAPI) attached programmable logic. With the CAPI [59], the FPGA accelerator appears as a coherent CPU peer over the I/O physical interface, thus accessing to a homogeneous virtual address space spanning the CPU and the accelerator, as well as a hardware-managed caching system. The advantages are clear: a shorter software path length is required compared to the traditional I/O model. The interface is implemented through two hardware sub-blocks: the Power Service Layer (PSL), and the Accelerator Functional Unit (AFU), i.e., the silicon block implementing the acceleration logic. The PSL block contains the hardware resources that maintain cache coherency, acting as a bridge between the AFU and the main CPU. An Interrupt Source Layer (ISL) is available in order to create an access point to the AFU for the software layer. On the CPU side, the Coherent Attached Processor Proxy (CAPP) acts as a gateway for serving the requests coming from and directed to the external AFU. Although the implementation of PSL and ISL units consumes resources on the FPGA (i.e., Flip-Flops, LUTs, RAM blocks, etc.), the adoption of high-end reconfigurable devices leaves enough space to implement complex hardware logic modules. In this context, OPERA leverages on the last Intel products (Arria 10 System-on-Chip – SoC [56]), which supports IEEE-754 Floating Point arithmetic through newly integrated DSPs. Beside pure reconfigurable logic, the Intel Arria 10 SoC features the second-generation dual-core ARM Cortex-A9 MPCore processor, which is integrated into the hard processor system (HPS) to obtain more performance, flexibility, and security with respect to the previous generation or equivalent soft-cores. On-die ARM cores allow the seamless integration of the reconfigurable logic with a general purpose elaboration pipeline, and in a broader perspective its integration in the complex DC architecture.

Scalability of the platform is further ensured by the adoption of on-board optical links. OPERA opted for standard Quad Small Form-factor Pluggable (QSFP) modules that permit up to 40 Gb/s of bandwidth, physically configured either as a single 40 Gb/s link or split into four independent 10 Gb/s links. This unprecedented level of flexibility allows for supporting a wide range of topologies. The hardware design issues and constraints are abstracted away and automatically handled by the Intel OpenCL compiler, leaving the software programmer to deal only with specific algorithms of interest. The compiler allows optimising high-level code (e.g., C-based code) enabling OpenCL channels (an OpenCL language construct) to be used for kernel-to-kernel or I/O-to-kernel data transfers at high bandwidth and low latency. Channels are also used to implement an application program interface

(API) intended for the host to communicate with the hardware accelerator, generally mapping the PCI Express interconnect.

4.2 Workload decomposition

To take full advantage of hardware specialisation, a mechanism to automatically assign tasks must be put in place. To this purpose, OPERA mostly exploits the *microservice* model. It has recently emerged in the Cloud community as a development style which allows building applications composed by several small independent but interconnected software modules [58]. Each module runs its own processes and communicating with others by means of a lightweight mechanism, typically consisting of an HTTP-based REST API, resulting in an asynchronous, shared-nothing, highly scalable software architecture. In order to automatise the deployment phase of such microservices-based Cloud applications, an ad-hoc "descriptor" is used. It allows the abstraction of the different software components and their relationships. To this end, OPERA leverages on the OASIS TOSCA [60] standard (Topology and Orchestration Specification for Cloud Applications), which enables the portability of Cloud applications and services across different platforms, and that has recently been extended to support Linux containers. TOSCA provides meta-model expressed with an XML-based language. It consists of two main parts: *i*) a topology template – a graph in which typed nodes represent service's components and typed relationships connect nodes following a specific topology; and *ii*) plans – work-flows used to describe managing concerns.

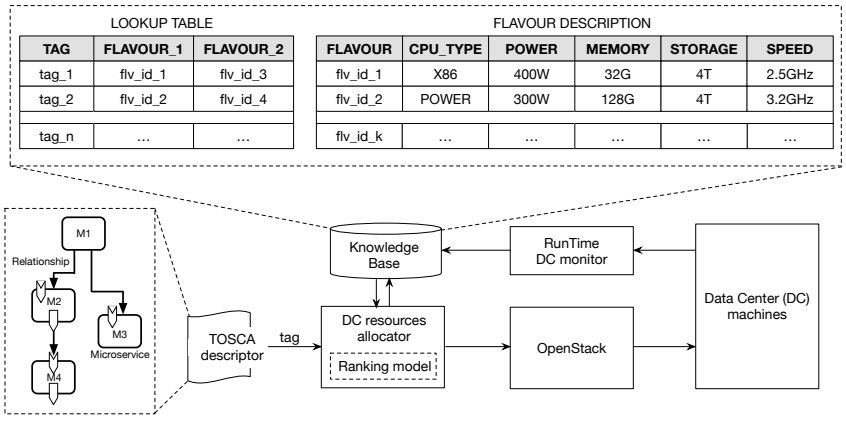

Fig. 4 The deployment chain used in OPERA to allocate resources within the heterogeneous data center. On top, the knowledge base – KB is depicted (table with list of nodes is not represented).

In OPERA, the flexibility offered by the TOSCA application descriptor is exploited to create hooks used to correctly assign microservices to the most suitable

hardware resource for their execution. Although, it is our intention to design and develop a solution that can be used in different context (i.e., not locked to a specific vendor existing platform), OPERA selected one popular platform as a reference: the OpenStack Cloud orchestration system. With regards to the integration of additional components to the OpenStack system, our solution takes into consideration a 2-steps allocation strategy:

- *Phase–1*: deploy the application components on the most suitable platform (i.e., by choosing the better processor architecture, amount of memory, storage space, etc.), with the aim of maximising the energy efficiency;
- *Phase–2*: periodically rescheduling (i.e., migrating) the application components on the most suitable platform if different load/efficiency conditions arise. For instance, a web front-end previously running on an ARM-based server can be moved on a X86_64 machine if the load of the ARM machine exceeded a threshold and/or the number of requests to the front-end increased.

In this regard, we refer to phase–1 as a *static deployment action*, while we talk of *dynamic (re-)scheduling* of the application components in phase–2. To perform such actions, the Cloud orchestrator needs to match microservices with the most suitable hardware resources based on the indication collected in a Knowledge Base (KB), and to monitor the status of the infrastructure.

Since energy consumption ultimately depends on the power consumption of the host machines in the DC, the enhanced Cloud orchestrator (we can refer to it as the Efficient Cloud Resources Allocation Engine – ECRAE) exploits a simple but rather effective power-model to select the host for execution. The power-based model becomes necessary to implement a greedy allocation strategy (Phase–1). Greedy allocation strategy does not ensure an optimal allocation for the whole set of microservices, thus, a further optimisation process is required (Phase–2). Here, a global optimisation algorithms is used to re-schedule all the allocated components with the objective of globally reducing the power (energy) consumption.

4.2.1 Efficiency model

The most critical element in the selection of the actual resource for executing a microservice is the model used to rank the machines belonging to the DC infrastructure. One point to keep into consideration is the relation between energy (E) and power consumption (P). The power P refers to the "instantaneous" energy consumed by a system and generally varies over the time. Based on that, the energy consumed by a system can be computed as the integral of the power consumption function on a given period of time: $E = \int_{t_0}^{t_1} P(t)dt$. The power consumed by a server machine depends on several factors; however, it can be assumed that it is mostly influenced by the consumption of the main hardware components (i.e., CPU, memory, storage and network interface). Power consumption of the CPU and the memory depends on how much the software running on that node stresses these components. Since the load generally changes over time, thus also the power consumed by the

CPU and memory (as well as other components) changes. Also, power consumption in idle state is critical. In literature [3, 6, 62] has been well documented that a conventional server machine, especially if not properly designed, may consume up to 65% of the maximum power consumption in the idle state. It is also worth noting that given the nature of a service, it becomes difficult to foreseen for how much time it will last. For instance, it is not possible to define the amount of time for which a database should run, since it is expected to be accessible unless a failure arises.

To tackle the above mentioned challenges, we elaborated a simple but still effective model for ranking nodes in the infrastructure. By profiling the behaviour of various microservices, we assume that each software component increases the CPU and memory load for a given quantity. Such quantity (C_l – represents the CPU load increase expressed as a percentage, M_l – represents the memory load increase expressed as a percentage) is measured as the average increase generated by the execution of that software component using the host machine in different working conditions. The following equation allows to emit a score value (R) for a given node:

$$R = \{\alpha\, C_l + (1-\alpha)\, M_l\}\; P_{avg} \tag{2}$$

The score R is the weighted measure of the current power consumption P_{avg} of the node (the power weighted value is biased by the power consumption of the node in idle state, so that the P_{avg} value is given by the power consumption in idle state incremented by the fraction due to the machine load); where the weight is expressed by a linear combination of the current CPU load (C_l) and the memory load (M_l). The α parameter allows to tune the power weight, considering the eventual imbalance between CPU and memory load factors. For instance, setting $\alpha = 0.25$, the load on the memory would be equal to 75%. Power consumption (P_{avg}) is obtained as a measure of the average power consumption of the host platform in different working conditions. Averaging the power consumption allows to capture the typical power profile of the host system. Such value is read directly from the Knowledge Base, thus it must be periodically updated to better reflect real machine behaviour. Such mechanism requires the availability of a hardware power monitor and an interface to query it. As part of the deployment mechanism, the envisioned high-density server enclosures are equipped with such monitoring infrastructure, to ease the action of the ECRAE system.

4.2.2 Static allocation strategy

TOSCA provides a hierarchical description of a generic Cloud application, which is at the basis of the mechanism used to trigger the "static" allocation strategy. For each element of the hierarchy, the set of scripts to manage the installation and the main functionalities exposed by the component are provided. The last element in the hierarchy is represented by host features. In OPERA we propose to use a tag to describe the affinity of the software components and the host features. Such affinity is representative of a possible configuration that is evaluated as the most suitable for

Algorithm 1: Static allocation strategy

Input: Knowledge Base (KB)
Output: DC machine where to execute the microservice

$T_c, T_m, \alpha \leftarrow get\_next\_task()$
$tag \leftarrow get\_tag()$
$a_1, a_2 \leftarrow get\_affinity()$
$N_1[\,] \leftarrow get\_node\_list(a_1)$
$N_2[\,] \leftarrow get\_node\_list(a_2)$
$score[\,] \leftarrow \emptyset$
for ***each*** n_{id} ***in*** N_1 **do**
    $C_l, M_l, P_{avg} \leftarrow get\_node\_status(n_{id})$
    $R \leftarrow \{\alpha\, C_l + (1-\alpha)\, M_l\}\ P_{avg}$
    $score[\,] \leftarrow add\_entry(R, n_{id})$
$score[\,] \leftarrow sort(score[\,])$
$sel\_node\_id\_best_1 \leftarrow select\_min\_score(score[\,], R)$
$go \leftarrow$ **True**
while $go =$ ***True or*** $score[\,] = \emptyset$
do
    $sel\_node\_id_1 \leftarrow select\_min\_score(score[\,], R)$
    if $sel\_node\_id_1(C_l) + T_c < 0.98$ ***and***
    $sel\_node\_id_1(M_l) + T_m < 0.98$ **then**
        $go \leftarrow$ **False**
    else
        $score[\,] \leftarrow remove\_entry(R, sel\_node\_id_1)$
if $go =$ ***False*** **then**
    $sel\_node\_id\_best_1 \leftarrow sel\_node\_id_1$
for ***each*** n_{id} ***in*** N_2 **do**
    $C_l, M_l, P_{avg} \leftarrow get\_node\_status(n_{id})$
    $R \leftarrow \{\alpha\, C_l + (1-\alpha)\, M_l\}\ P_{avg}$
    $score[\,] \leftarrow add\_entry(R, n_{id})$
$score[\,] \leftarrow sort(score[\,])$
$sel\_node\_id\_best_2 \leftarrow select\_min\_score(score[\,], R)$
$go \leftarrow$ **True**
while $go =$ ***True or*** $score[\,] = \emptyset$
do
    $sel\_node\_id_2 \leftarrow select\_min\_score(score[\,], R)$
    if $sel\_node\_id_2(C_l) + T_c < 0.98$ ***and***
    $sel\_node\_id_2(M_l) + T_m < 0.98$ **then**
        $go \leftarrow$ **False**
    else
        $score[\,] \leftarrow remove\_entry(R, sel\_node\_id_2)$
if $go =$ ***False*** **then**
    $sel\_node\_id\_best_2 \leftarrow sel\_node\_id_2$
if $score[sel\_node\_id\_best_2] < score[sel\_node\_id\_best_1]$ **then**
    return $sel\_node\_id\_best_2$
else
    return $sel\_node\_id\_best_1$

the execution of the specific component (e.g., a big-memory tag could be used to represent the configuration of a big memory machine, which is well suited for in-memory database operations). The correspondence between the affinity expressed in the TOSCA descriptor and the node configurations is provided by the Knowledge Base. Here, for each configuration, the list of nodes which provide that configuration is also available, along with the actual CPU load, memory load and the average power dissipation. Once the node that better fits with the ECRAE policies has been selected, the corresponding full configuration is used to replace the affinity element in the TOSCA descriptor. The result of such selection process for each application component is a fully-compliant TOSCA description file, which can be transformed into a OpenStack compliant description. Interestingly, although the affinity with a specific host configuration is a static information provided in the TOSCA description, OPERA aims at finding and integrating a mechanism able to allow the orchestration system to "automatically learning" which is the best affinity mapping, as a future investigation direction. To the purpose of fast retrieving information, the KB is organised as a (relational) database. Specifically, three tables provide the required information. One table is used to map tags with a list of possible machine configurations. A second table provides the information of each node matching the specific configuration. Finally, a third table provides the current CPU and memory load for each node. This information is updated by an external component (e.g., Ceilometer module in the OpenStack, Carbon/Graphite, etc.). All these information then are combined into a simple ranking model (see equation 2).

The algorithm used by ECRAE is provided as a pseudo-code (see Algorithm–1). The first step is to extract the information related to the application component to allocate (lines 1–2). This information regards the increment in terms of CPU and memory loads (as a percentage), the tunable α parameter, and the affinity tag. Given the affinity tag, in line 3 the corresponding configurations (affinities) are extracted. We assume that the first configuration (a_1) represents the best match with the requirements of the application component; however, an alternative configuration can be exploited (a_2). Given the effective configurations, the algorithm extracts the list of nodes in the data center that have those configurations (lines 4–5), then it creates an empty list to associate to each node a corresponding R score. In lines 7—10, a loop is used to create such list (the ranking model described in Section 4.2.1), and in line 11 the list is sorted. The first element (line 12) is the candidate node providing the lowest power increase, but it is needed to verify if the corresponding increase in the CPU and memory loads are acceptable. To this end, the algorithm analyses all the available nodes (lines 13–21). If no one of the nodes can be further loaded (a threshold of 98% is assumed), the algorithm maintains the initial candidate solution. Similarly, in lines 22–40, a candidate solution is searched in the list of alternative configurations. Finally, in lines 41–44, the best ever node is returned. Given this node, the corresponding exact configuration is substituted in the TOSCA descriptor, and the application is allocated through the OpenStack environment (i.e., the Linux container or the required virtual machine are instantiated on the selected node).

For instance, lets consider two nodes, each of them belonging to one of the flavours associated to a given affinity tag. For the sake of simplicity we can as-

sume two X86_64 nodes, each in the idle state, but with different average power consumption: we assume $node_1$ consuming up to 100 W (i.e., assuming 65% of idle power consumption that is equal to 65W), $node_2$ consuming up to 130 W (i.e., assuming 65% of idle power consumption that is equal to 84.5 W). Lets assume to schedule two tasks loading the nodes by 45% each (i.e., the CPU load and memory load are assumed equals to 0.45, using $\alpha = 0.5$). Given this premise, the basic allocation policy (i.e., assigning the task to the less loaded node) leads to a higher power consumption, as reported in figure 5.

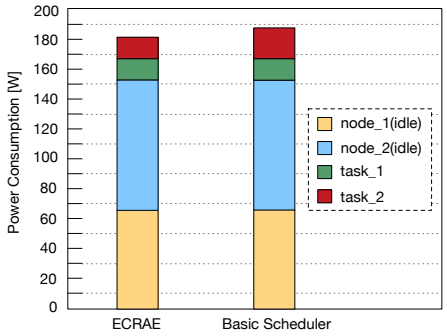

Fig. 5 Example of the power (energy) saving provided by the static allocation strategy: two different nodes are considered respectively with their idle power consumption.

In fact, when the first task is selected, the two nodes are in the idle state and both the strategies (basic and the one implemented by ECRAE) allocate the task to $node_1$. At this point, the power consumption of $node_1$ increases up to 80.75 W, with an overall power consumption, for the two nodes, equals to 165.25 W. On the other hand, the second task is allocated differently. ECRAE ranks the node depending on their weighted power consumption, thus selecting $node_1$ also for the second task (although $node_2$ is less loaded). This provides further 15.75 W of power consumption (increasing the load up to 90%). Conversely, basic allocation strategy selects the node with the lowest load, leading $node_2$ to be selected. In that case, the execution of second task on such node provides 20.475 W of power consumption, leading to an overall power consumption of 185.72 W. Although the power saving for the two nodes is around 5 W, if we consider such saving on a large slice of the DC servers, we obtain a huge improvement in terms of energy saving.

4.2.3 Dynamic allocation strategy

Static workload allocation exploits an aggressive greedy strategy that does not ensure global optimal allocation of the resources. To guarantee such optimality, an instance of a global scheduling problem (i.e., Bin Packing Problem – BPP) must be solved. Workload scheduling is a well-known optimisation problem that fits in the

complexity class of NP-Hard problems. It can be formulated as follows: *Given a set of different objects (VMs or containers), each with a volume S_i (i.e., the amount of CPU and memory used), the objective is to assign as much as possible objects to a bin (i.e., a server machine) that as a finite volume V_j (i.e., a certain amount of CPU and memory offer)*. In the context of workload scheduling, the problem requires the minimisation of the number of running machines (i.e., the number of used bins), and also the whole power consumption of the data center (i.e., an heuristic should try to consolidate as much as possible the workload on the minimum number of active hosts). Among various algorithmic solutions, evolutionary-based (and in particular the Particle Swarm Optimisation) techniques allow to quickly solve large instances of this problem.

Particle Swarm Optimisation (PSO) is a population-based stochastic meta-heuristic developed by Kennedy and Eberhart in 1995 to optimise multi-modal continuous problems. In PSO, a group of independent solutions (termed as particles) are used to sample the search space and discover the optimal solution. The state of such group of particles is evolved over time, by updating particles' position within the multi-variable search space. Passing from one position in a given instant of time to another is made by taking into account particles' velocity. The velocity and the position of the particles are taken care by two components, which are described as two factors incorporating a form of distributed intelligence:

- *Cognitive factor*: encodes the information regarding the history of the best position assumed by the particles at certain moment in time.
- *Social factor*: encodes the information relating to the history of the best position assumed by the neighbourhood of the particle at certain moment in time.

These two factors are used to adapt the velocity of the particles in such way it can steer the position towards the optimal solution. PSO does not make use of operators devoted to combining solutions belonging to the same population. On the contrary, the social factor allows to incorporate the knowledge collected by other particles. The topology of the neighbourhood influences the behaviour of the heuristic, although the entire set of particles is used as the neighbourhood (i.e., lattice model). The lattice model also has the advantage of keeping the number of operations used to determine the absolute best position low.

The PSO algorithm is a key component of the ECRAE, since it allows to periodically redistribute the workload on the DC resources, towards their efficient exploitation. At the basis of this periodic imbalance reduction strategy, there is the possibility of migrating application components from one machine to another. Traditional virtual machines can be transparently migrated, but their associated overheads are not acceptable for an effective implementation of the microservices model. To overcome this limitations, Linux containers must be put in place. Linux containers are not designed to be migrated, but such feature becomes important to fully unleash their potential. In the following Section we discuss the way adopted in OPERA to solve this challenge.

4.3 Workload Migration

If microservices are in use (i.e., containerised application components), and the microservices have been designed to be scalable and resilient, it is possible that scaling up (i.e., creating more instances) or scaling down (i.e., destroying instances) of a given microservice is the best way to balance the load. In other cases, creating or destroying containers is not possible, and we must resort to migration. Virtual machines are more heavy-weight example of an execution context, and the price for instantiating new VMs may be considerably higher than migrating an existing one. On the contrary, Linux containers are an example of a light-weight execution context for the microservices model. In the OPERA project, not only we look at how to implement container migration, but also how to perform that in a heterogeneous data center. First, let us list and explain the options that do not make sense. We can then deal with the remaining options as viable, and look at which ones we are focusing on.

In the case of machines with different ISAs (but still general purpose CPUs) VM migration between two machines may have merit. Today, it is possible to migrate a VM to a server exposing a different ISA by employing a binary translation mechanism (such as QEMU), which can emulate the target ISA and translate from the source ISA through software functions. This kind of emulation is at least an order of magnitude slower than running native code, which means it should not be used in performance-critical situations. Due to the amount of software running, it also consumes more energy than the native equivalent, and is therefore not efficient from several perspectives.

Migrating workloads to accelerators (such as GPGPUs) is a possible avenue that may show results in the future. However, nowadays, due to many restrictions and architectural specialisation, accelerator are not able to manage the execution of a whole VM or container. In fact, such technologies relies on features available on general purpose processors, and they are mostly developed around the widely adopted X86_64 architecture. Although in OPERA the objective is to seamlessly access to accelerators (e.g., GPGPUs or even to FPGAs), their usage is limited to the offloading of computational-heavy functions. Here, migration in an heterogeneous context means focusing on multiple ISAs and different types of compute resources, but still relying on general purpose processors. For these reasons it does not make sense to migrate a virtual machine or container to such accelerators. It is likely that much more efficient methods of moving a workload to such compute resources will exist. Furthermore, in the case of FPGAs, the acceleration is mainly a static mapping between a computational-heavy function and a dedicated circuit. Even if we are going to consider the case of "high level synthesis" languages such as OpenCL, there is not yet a clear path for migrating general applications (or portions thereof) to such wildly different compute resources. Further, although techniques to dynamically reconfigure the FPGA devices exist and can be used, their overhead remains too high to justify their implementation on a large scale.

4.3.1 Container Migration

Container migration (and process migration as the generalised case) is a more relevant problem for which we can develop a solution. Containers gain importance due to the advent of microservices, in which application are decomposed, and each component is isolated from the others by means of containers. Some containers run services that cannot easily be replicated or scaled (such as in-memory databases), and the cost to migrate such containers is likely to be less than attempting replication. Moreover, container migration is performed at the system level, which means the application does not need to be aware of how to migrate itself, nor that the migration is even taking place. Thus we may provide migration support for such containerised applications that do not contain support for scaling or replication.

There are several popular implementations of containers on the Linux operating system, such as Docker, LXC/LXD, and Runc. Those container implementations rely on Checkpoint-Restore in Userspace (CRIU) tool for checkpointing, restoring and migrating the containers. Although CRIU relies heavily on advanced features found in the Linux kernel, it does not require any modifications of the kernel itself and it is able to perform checkpoint and restore operations entirely in userspace. At the basic level, CRIU allows freezing a running application and checkpointing it as a collection of files. These files can be later used for restoring the application and continuing execution from the exact point where it was frozen. This basic checkpoint-restore functionality enables several features such as application live migration, application snapshots, remote application analysis and remote debugging. Any flavor of Linux containers can be abstracted as a process tree along with additional properties required for process isolation and fine grained resource management. These processes may have access to various virtual and pseudo devices. CRIU is capable to snapshot the state of the entire process tree as well as the state of the virtual and pseudo devices the processes in the process tree are using. In addition, the properties required for process isolation and fine grained resource management are saved and become an integral part of the container state snapshot.

4.3.2 Comparing post-copy and pre-copy migration techniques

The container state snapshot contains several components that describe process state, open file descriptors, sockets, Linux namespaces, state of virtual and pseudo devices. Yet, all these objects are small and can be easily migrated between different hosts with negligible latency. The part of the container state requiring most of the storage capacity for a snapshot or most of the network bandwidth for a migration is the memory dump of the processes that run in a container. For the case of the container migration, the amount of the memory used by the applications running inside the container defines the time required to migrate the container, as well as the downtime of the application.

The simplest and naive implementation of container migration is as follows: *i*) freeze all the processes inside the container; *ii*) create a snapshot of the entire con-

tainer state, including complete memory dumps of all the processes; *iii*) transfer the container state snapshot to the destination node; and *iv*) restore the container from the snapshot. In this case, the time required to migrate the container and the downtime of the application running inside it are equal and both these times are proportional to the amount of memory used by the processes comprising the container. The application downtime during migration may be decreased with one or more round of memory *pre-copy* before freezing the container. With iterative memory pre-copy, container migration time is slightly longer that in the simple case, but the actual downtime of the application is significantly smaller in most cases. However, such approach may not work for applications with rapidly changing memory working set. For such applications, the amount of modified memory will always be higher than the desired threshold, and therefore the iterative pre-copy algorithm will never converge.

An alternative approach for reducing the application downtime is called *post-copy migration* (this is also the approach adopted in OPERA). With post-copy migration, the memory dump is not created and memory contents is transferred after the application is resumed on the destination node. The primary advantage of post-copy migration is its guaranteed convergence. Additionally, post-copy migration requires less network bandwidth than iterative pre-copy migration, since the memory contents are transferred exactly once. The migration time in this case is small because only the minimal container state snapshot is transferred before the execution is resumed on the destination node. The application downtime is almost as small as the migration time, however, immediately after migration the application will be less responsive because of the increased latency for memory access (this initial low responsiveness is generally tolerated in Cloud applications).

4.4 Optimizing virtual memory management

Also, finding the correlation between using different compute resources and runtime help the development of a methodology for performance estimation, in terms of run-time and energy consumption. Accurate and fast methods for performance estimation will be beneficial, for example, in workload management and dynamic allocation of resources. As the number of huge memory pages that can be allocated and the number of threads are limited in increasing performance perspective, then allocating these limited resources between applications that run on the same system requires to have some model to find the best allocation of these resources, to get the best performance. In modern computing platforms and data centers, the RAM size is not the main performance bottleneck (and energy consumption source), and modern computing platforms can support terabytes of RAM. But, increasing only the RAM size does not increase address translation table (TLB), which is used in modern CPUs to quickly access memory locations. Because TLB capacities cannot scale at the same rate as DRAM, TLB misses and address translation can incur crippling performance penalties for large memory workloads. TLB misses might de-

grade performance substantially and might account for up to 50% of the application run-time [61]. So, increasing only RAM size will not improve the performance for some applications (especially memory intensive ones), that suffer from TLB misses. Therefore, using huge memory pages can save some of these penalties by the fact that using TLB entries of huge pages covers much more memory space than when using the same number of TLB entries of base pages.

There are two main challenges in running the profiling work on modern computing platforms. The first is that these platforms are designed to run multiple tasks on multiple cores, and then we should profile only the running work without being affected by other tasks that run on the same system, or on the same core. The second challenge, is that developing an estimation model requires few different samples of run-time for different page-walks or threads, but getting diversity in TLB page walks for the same workload is more challenging, in terms of controlling the page walks or the allocated huge pages.

Hardware performance counters are set of special-purpose registers built into modern microprocessors to store the counts of hardware-related activities within computer systems. Advanced users often rely on those counters to conduct low-level performance analysis or tuning. The main counters and hardware events we are interested in using for analysing applications' behaviour and drawing an evaluation model are: *i*) the DTLB_LOAD_MISSES:WALK_DURATION, and *ii*) DTLB_STORE_MISSES:WALK_DURATION (this are available on X86_64 processors, but similar can be exploited on different architectures); which respectively count the total cycles Page Miss Handler (PMH) is busy with page walks for memory load and store accesses. The majority of the processors also implement performance counters collecting information regarding the power and energy events, which are of interest to find a correlation between the application performance and the energy efficiency of the hardware in use. An example of such performance counters is represented by the RAPL interface available on the Intel X86_64 processors. RAPL is not an analog power meter, but rather uses a software power model. This software power model estimates energy usage by using hardware performance counters and I/O models. Based on the analysis of the GUPS benchmark (this is representative of memory intensive workloads found in DCs) running on two reference machines, the impact on the memory subsystem can be analysed. Generally, plotting the application run-time as a function of DTLB load page walks provides a simple linear model. Such simple model offers more space for understanding the application behaviour and for optimising the memory access pattern (we can assume that page walks have a linear overhead on run-time). Such conclusion is confirmed by the analysis of the energy consumption as a function of DTLB load page walks. Again a linear model is enough to explain collected data, showing large opportunity to optimise the application energy impact. Although enough accurate to capture application behaviour in most of the cases, more complex model (e.g., quadratic one) should be used to limit the evaluation error.

5 Ultra-low power connected cyber-physical systems

Cyber-Physical Systems (CPS) are becoming an essential part of modern large scale infrastructures, such as in the case of Cloud data centers. CPS provide enough computing and storage capabilities to pre-process captured data, before streaming them in the Cloud back-end. Also, they embed different sensors and actuators, so that they make easy to remotely interact with the surrounding physical environment. However, to further enlarge CPS adoption, more energy efficient technologies must be put in place, as well as a more effective way of exploiting back-end capabilities for processing captured data. To this end, OPERA provides, not only a highly integrated design, but also a mechanism to effectively offload more computational intensive tasks on high-performance accelerated servers.

5.1 Accelerating smart vision applications

Video surveillance is one of the most interesting application fields for smart connected devices. In OPERA, "intelligent" cameras are used to monitor urban traffic aiming at recognising potential situations of risk. Such kind of application covers multidisciplinary fields, spanning computer vision, pattern recognition, signal processing, and communication. The complexity of the application is high, and when it has to be performed in a time-constrained manner, the situation is exacerbated. Meeting the requirements for such applications means implementing advanced hardware systems, albeit generally with a low energy efficiency. From this viewpoint, integration of functionalities in the form of dedicated hardware modules is considered as a technological key feature to increase CPS efficiency. Video surveillance also exposes large parallelism to the hardware: processing functions are applied to (groups of) pixels in parallel.

For this reason, OPERA design adopts a highly-parallel architecture based on an ultra-low power (ULP) manycore solution designed to operate with very low supply voltage and currents. Specifically, it exploits energy efficient cores (EE-cores) which can accelerate several operations in hardware, as well as exploit an energy-optimised instruction set architecture. With such features, the envisioned computing layer can perform operations only requiring few pJ of energy. In addition, a dedicated image processing unit allows performing complex operations, such as moving object detection and image/video compression, at a low energy consumption when compared with standard software implementations. Also, OPERA aims at further improving the performance/power ratio by integrating HW/SW components to accelerate convolutional neural networks (CNNs). CNNs allow, with a limited increase in the used resources, to improve the identification of classes of objects on the scene (this task is the basis for any video monitoring application). However, every time the scene to analyse requires more computational capabilities with respect to that available on the CPS, the analysis task can be effectively offloaded on a remote high-performance server. Thorough a dedicated API, the CPS can access to a Cloud

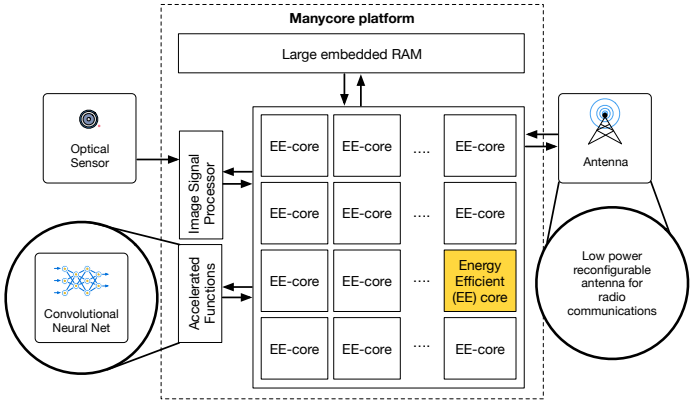

Fig. 6 OPERA Cyber-Physical System architecture: an ultra-low power manycore processor with acceleration function for CNNs is directly attached to the camera sensor and to the reconfigurable communication antenna.

(micro)service: here, more complex algorithm can be run and efficiently accelerated using GPGPUs or FPGA boards.

The complete CPS design (see Figure 6) envisages a low power reconfigurable radio-frequency (RF) communication interface. This kind of communication interface is of particular interest for video surveillance and monitoring applications where the environmental context is particularly critical, such as in the case of mountain roads where connectivity is not reliable. To deal with this issue, our reconfigurable RF module will be capable of adapting the transmission to the best channel/protocol features. Since RF transmission is generally power-hungry, we will design the RF interface with very low power components.

6 The real-life application context

OPERA aims at providing technology demonstration on improved energy efficiency, scalability, and computational performance by resorting to three real-world applications.

6.1 Virtual desktop infrastructure

The purpose of this use case is to demonstrate how the set of technologies taken into consideration by OPERA, allows to make data centers more scalable, and energy efficient. The idea is to exploit the “as-a-service” model to provide a virtual desktop infrastructure (VDI), i.e., a remotely accessible desktop environment. Users are increasingly demanding access to their applications and data anywhere and from

any device. The rapid growth of "nomadic" workers who roam from one computer/device to another leads organisations to provide access to the users' desktop experience at any computer in the workplace, effectively detaching the user from the physical machine. Virtualization is at the basis of this process: employees can access their applications and data very safely over a network, and the risk of data loss is minimised. On the other hand, such practice allows IT departments also to save costs by consolidating services on physical resources (server machines hosted in private or public DCs).

To address this challenge, OPERA implements a solution based on the open-source framework OpenStack. Specifically, OpenStack components such as Cinder and Ceph file system, are used to cover block storage needs for virtual machines and containers. Given the low-latency requirements of this use case, network management is also concerned. To this end, OPERA exploits the flexibility furnished by Neutron. In addition, network latency will be kept low by leveraging on a more powerful remote desktop protocol w.r.t. the traditional protocols (e.g., VNC, RDP, etc.). Finally, to keep as low as possible the overhead of the software virtualization layer, a mechanism based on the KVM hypervisor and containerisation is put in place to run lightweight virtual machines on low power servers.

6.2 Mobile data center on truck

OPERA intends to deliver mobile IT services for the Italian agency called Protezione Civile. IT services, such as forecasting and risk prevention, contrasting and overcoming emergencies, are delivered through a truck (operated by partner CSI Piemonte) equipped with electronic instruments which allow: *i*) creating a satellite communication link; *ii*) acquiring images and videos of area surrounding the truck; and *iii*) processing, temporary archiving and transferring acquired data (videos and images). Images and videos are captured using a drone equipped with cameras and wireless connectivity. The use of a drone is helpful also in the case of dangerous or difficult access to the target site. Once acquired, images and videos are then processed on the truck. Data processing consists of these two steps: *i*) the arrangement of the videos by deleting not useful parts, adding comments, etc; and *ii*) the creation of *orthoimages* for their subsequent comparison with others archived. An orthoimage is a normalised image with respect to a given reference framework, and the creation of orthoimages generally represent a compute-intensive task. For instance, 20 minutes of flight yield 300 photos that require approximately 15 hours to be processed with a standard X86_64 machine (having a resolution of 10^{-2} m. Moving from a standard computer to a high-density but low power server equipped with an FPGA accelerator enables OPERA to greatly speedup the processing task: the elaboration of the same set of photos can be completed in roughly 30 minutes, while keeping low the impact on the power source of the truck (currently, a gasoline power generator).

6.3 Road traffic monitoring

OPERA foresees a growing interest in using remotely-controlled CPS for traffic monitoring purposes in urban and rural contexts. Deploying ultra-low power CPS equipped with effective video processing and wireless communication capabilities, makes possible to monitor large geographic areas in a more energy-efficient manner. For instance, it becomes possible to quickly detect accidents or any situation of risk and communicate alerts (eventually also to vehicles). To this end, collected and pre-processed data are transferred to low power servers located in a remote data centers for further analysis and proactive actions intended to reduce such risk situations.

7 Conclusion

In this chapter, we have presented hardware and software technologies, as well as their integration into a fully functional infrastructure covering the whole computing continuum. Such mix of technological solution are still under development, within the context of the OPERA H2020 European project. This project aims at improving the energy-efficiency of current Cloud computing infrastructures by two order of magnitude when compared with current state-of-the-art systems. To accomplish with this challenging objective, the integration of several advancements on the data center side, as well as on the end nodes of the Cloud is envisioned. Particular attention to the integration of (ultra-)low power high performance technologies is of primary interest for the project.

Specifically, OPERA foresees to gain efficiency on the data center side, by proposing a modular, high-density server enclosure equipped with small low power server boards, and accelerator cards. To maximise the efficiency, FPGA devices will be used in the accelerator boards to provide acceleration for specific kernels as well as low latency connectivity towards POWER nodes. In addition, to fully exploit this wider heterogeneity, applications leverage on a modular architectural style (microservices) that allows to better scale. On the other hand, Cloud end-nodes (i.e., CPS) are made less power-hungry by integrating manycore processing elements with dedicated hardware accelerated functions and reconfigurable wireless communication interfaces. To assess the feasibility of the envisioned platform, OPERA aims at testing its solution on three real-world applications, albeit the results carried out in the project are of interest for a broader community.

Acknowledgement

This work is supported by the European Union H2020 program through the OPERA project (grant no. 688386).

References

1. http://www.operaproject.eu
2. Filani, David and He, Jackson and Gao, Sam and Rajappa, Murali and Kumar, Anil and Shah, Pinkesh and Nagappan, Ram, "Dynamic Data Center Power Management: Trends, Issues, and Solutions.", *Intel Technology Journal*, vol. 12, issue 1, 2008.
3. Barroso, Luiz André and Hölzle, Urs, "The Case for Energy-Proportional Computing", *Computer*, vol. 40, pp.33–37, 2007.
4. Barroso, Luiz André and Clidaras, Jimmy and Hölzle, Urs, "The datacenter as a computer: An introduction to the design of warehouse-scale machines", *Synthesis lectures on computer architecture*, vol. 8, no. 3, pp. 1–154, Morgan & Claypool Publishers, 2013.
5. Greenberg, Albert and Hamilton, James and Maltz, David A and Patel, Parveen, "The cost of a cloud: research problems in data center networks", *ACM SIGCOMM computer communication review*, vol. 39, no. 1, pp.68–73, ACM, 2008.
6. Fan, Xiaobo and Weber, Wolf-Dietrich and Barroso, Luiz Andre, "Power provisioning for a warehouse-sized computer", *ACM SIGARCH Computer Architecture News*, vol. 35, pp. 13–23, ACM, 2007.
7. Pearce, Michael and Zeadally, Sherali and Hunt, Ray, "Virtualization: Issues, security threats, and solutions", *ACM Computing Surveys (CSUR)*, vol. 45, no. 2, pp. 17, ACM, 2013.
8. Srikantaiah, Shekhar and Kansal, Aman and Zhao, Feng, "Energy aware consolidation for cloud computing", *Proceedings of the 2008 conference on Power aware computing and systems*, vol. 10, 20008.
9. Vogels, Werner, "Beyond server consolidation", *Queue*, vol. 6, no. 1, pp. 20–26, ACM, 2008.
10. Kaur, Tarandeep and Chana, Inderveer, "Energy efficiency techniques in cloud computing: A survey and taxonomy", *ACM Computing Surveys (CSUR)*, vol. 48, no. 2, pp. 22, ACM, 2015.
11. Beloglazov, Anton and Abawajy, Jemal and Buyya, Rajkumar, "Energy-aware resource allocation heuristics for efficient management of data centers for cloud computing", *Future generation computer systems (FGCS)*, vol. 28, no. 5, pp. 755–768, Elsevier, 2012.
12. Murtazaev, Aziz and Oh, Sangyoon, "Sercon: Server consolidation algorithm using live migration of virtual machines for green computing", *IETE Technical Review*, vol. 28, no. 3, pp. 212–231, Taylor & Francis, 2011.
13. Van, Hien Nguyen and Tran, Frédéric Dang and Menaud, Jean-Marc, "Performance and power management for cloud infrastructures", *Cloud Computing (CLOUD), IEEE 3rd International Conference on*, pp. 329–336, IEEE, 2010.
14. Zhang, Qi and Zhu, Quanyan and Boutaba, Raouf, "Dynamic resource allocation for spot markets in cloud computing environments", *Utility and Cloud Computing (UCC), Fourth IEEE International Conference on*, pp. 178–185, IEEE, 2011.
15. Ardagna, Danilo and Panicucci, Barbara and Passacantando, Mauro, "A game theoretic formulation of the service provisioning problem in cloud systems", *Proceedings of the 20th international conference on World wide web*, pp. 177–186, ACM, 2011.
16. Quang-Hung, Nguyen and Nien, Pham Dac and Nam, Nguyen Hoai and Tuong, Nguyen Huynh and Thoai, Nam, "A genetic algorithm for power-aware virtual machine allocation in private cloud", *Information and Communication Technology*, pp. 183–191, Springer, 2013.
17. Li, Luqun, "An optimistic differentiated service job scheduling system for cloud computing service users and providers", *Multimedia and Ubiquitous Engineering, MUE'09. Third International Conference on*, pp. 295–299, IEEE, 2009.
18. Li, Kenli and Tang, Xiaoyong and Li, Keqin, "Energy-efficient stochastic task scheduling on heterogeneous computing systems", *Parallel and Distributed Systems, IEEE Transactions on*, vol. 25, no. 11, pp. 2867–2876, IEEE, 2014.
19. Ghribi, Chaima and Hadji, Makhlouf and Zeghlache, Djamal, "Energy efficient vm scheduling for cloud data centers: Exact allocation and migration algorithms", *Cluster, Cloud and Grid Computing (CCGrid), 13th IEEE/ACM International Symposium on*, pp. 671–678, IEEE, 2013.

20. Infrastructure – VMware, “Resource management with VMware DRS”, *VMware Whitepaper*, 2006.
21. Gürsun, Gonca and Crovella, Mark and Matta, Ibrahim, “Describing and forecasting video access patterns”, *INFOCOM, Proceedings IEEE*, pp. 16–20, IEEE, 2011.
22. Tirado, Juan M and Higuero, Daniel and Isaila, Florin and Carretero, Jesus, “Predictive data grouping and placement for cloud-based elastic server infrastructures”, *Proceedings of the 11th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing*, pp. 285–294, IEEE Computer Society, 2011.
23. Roy, Nilabja and Dubey, Abhishek and Gokhale, Aniruddha, “Efficient autoscaling in the cloud using predictive models for workload forecasting”, *Cloud Computing (CLOUD), IEEE International Conference on*, pp. 500–507, IEEE, 2011.
24. Chandra, Abhishek and Gong, Weibo and Shenoy, Prashant, “Dynamic resource allocation for shared data centers using online measurements”, *International Workshop on Quality of Service*, pp. 381–398, Springer, 2003.
25. Kumar, Anoop S. and Mazumdar, Somnath, “Forecasting HPC Workload Using ARMA Models and SSA”, *Proceedings of the 15th IEEE Conference on Information Technology (ICIT)*, pp. 1–4, IEEE, 2016.
26. Calheiros, Rodrigo N and Masoumi, Enayat and Ranjan, Rajiv and Buyya, Rajkumar, “Workload prediction using arima model and its impact on cloud applications’ qos”, *IEEE Transactions on Cloud Computing*, vol. 3, no. 4, pp. 449–458, IEEE, 2015.
27. Iqbal, Waheed and Dailey, Matthew N and Carrera, David and Janecek, Paul, “Adaptive resource provisioning for read intensive multi-tier applications in the cloud”, *Future Generation Computer Systems*, vol. 27, no. 6, pp. 871–879, Elsevier, 2011.
28. Beloglazov, Anton and Buyya, Rajkumar, “Adaptive threshold-based approach for energy-efficient consolidation of virtual machines in cloud data centers”, *Proceedings of the 8th International Workshop on Middleware for Grids, Clouds and e-Science*, vol. 4, ACM, 2010.
29. Chieu, Trieu C and Mohindra, Ajay and Karve, Alexei A and Segal, Alla, “Dynamic scaling of web applications in a virtualized cloud computing environment”, *e-Business Engineering, ICEBE’09. IEEE International Conference on*, pp. 281–286, IEEE, 2009.
30. Lim, Harold C and Babu, Shivnath and Chase, Jeffrey S and Parekh, Sujay S, “Automated control in cloud computing: challenges and opportunities”, *Proceedings of the 1st workshop on Automated control for data centers and clouds*, pp. 13–18, ACM, 2009.
31. Zhan, Shaobin and Huo, Hongying, “Improved PSO-based task scheduling algorithm in cloud computing, *Journal of Information & Computational Science*, vol. 9, no. 13, pp. 3821–3829, 2012.
32. Zhang, Hongli and Li, Panpan and Zhou, Zhigang and Yu, Xiangzhan, “A PSO-Based Hierarchical Resource Scheduling Strategy on Cloud Computing, *International Conference on Trustworthy Computing and Services*, pp. 325–332, Springer, 2012.
33. Liu, Zhanghui and Wang, Xiaoli, “A PSO-based algorithm for load balancing in virtual machines of cloud computing environment”, *International Conference in Swarm Intelligence*, pp. 142–147, Springer, 2012.
34. Crago, S.P. and Walters, J.P., ”Heterogeneous Cloud Computing: The Way Forward”, *IEEE Computer*, vol. 48, no. 1, pp.59–61, 2015.
35. M. Lavasani, H. Angepat and D. Chiou, ”An FPGA-based In-Line Accelerator for Memcached,” in *IEEE Computer Architecture Letters*, vol. 13, no. 2, July-Dec. 15 2014.
36. A. Putnam et al., ”A reconfigurable fabric for accelerating large-scale datacenter services,” *2014 ACM/IEEE 41st International Symposium on Computer Architecture* (ISCA), Minneapolis, MN, 2014.
37. A. Becher, F. Bauer, D. Ziener and J. Teich, ”Energy-aware SQL query acceleration through FPGA-based dynamic partial reconfiguration,” *2014 24th International Conference on Field Programmable Logic and Applications* (FPL), Munich, 2014.
38. A. Traber, et al., ”PULPino: A small single-core RISC-V SoC”, *RISC-V Workshop*, 2016.
39. N. Ickes, et al., ”A 10 pJ/cycle ultra-low-voltage 32-bit microprocessor system-on-chip”, *Proceedings of the ESSCIRC*, Helsinki, 2011.

40. S. Ciccia et al., "Reconfigurable antenna system for wireless applications," *Research and Technologies for Society and Industry Leveraging a Better Tomorrow* (RTSI), IEEE 1st International Forum on, Turin, 2015, pp. 111-116.
41. Andrew Canis, Jongsok Choi, Mark Aldham, Victor Zhang, Ahmed Kammoona, Tomasz Czajkowski, Stephen D. Brown, and Jason H. Anderson, "LegUp: An open-source high-level synthesis tool for FPGA-based processor/accelerator systems", *ACM Trans. Embed. Comput. Syst.*, vol. 13, no. 2, ACM, 2013.
42. J. Villarreal, A. Park, W. Najjar and R. Halstead, "Designing Modular Hardware Accelerators in C with ROCCC 2.0", *18th IEEE Annual International Symposium on Field-Programmable Custom Computing Machines*, pp. 127–134, IEEE, 2010.
43. A. Munshi, "The OpenCL specification", *IEEE Hot Chips 21 Symposium (HCS)*, pp. 1–314, 2009.
44. http://www.rapid-project.eu
45. Montella R., Ferraro C., Kosta S., Pelliccia V., Giunta G., "Enabling Android-Based Devices to High-End GPGPUs", *In: Algorithms and Architectures for Parallel Processing (ICA3PP) – Lecture Notes in Computer Science*, vol. 10048, Springer, 2016.
46. Evans, D., "The internet of things how the next evolution of the internet is changing everything", *CISCO White papers*, 2011.
47. Satyanarayanan, Mahadev and Bahl, Paramvir and Caceres, Ramón and Davies, Nigel, "The case for vm-based cloudlets in mobile computing", *IEEE pervasive Computing*, vol. 8, no. 4, pp. 14–23, IEEE, 2009.
48. Vaquero, Luis M and Rodero-Merino, Luis, "Finding your way in the fog: Towards a comprehensive definition of fog computing", *ACM SIGCOMM Computer Communication Review*, vol. 44, no. 5, pp. 27–32, ACM, 2014.
49. Willis, Dale F and Dasgupta, Arkodeb and Banerjee, Suman, "Paradrop: a multi-tenant platform for dynamically installed third party services on home gateways", *Proceedings of the 2014 ACM SIGCOMM workshop on Distributed cloud computing*, pp. 43–44, ACM, 2014.
50. Martins, Joao and Ahmed, Mohamed and Raiciu, Costin and Olteanu, Vladimir and Honda, Michio and Bifulco, Roberto and Huici, Felipe, "ClickOS and the art of network function virtualization", *Proceedings of the 11th USENIX Conference on Networked Systems Design and Implementation*, pp. 459–473, USENIX Association, 2014.
51. Patel, M and Naughton, B and Chan, C and Sprecher, N and Abeta, S and Neal, A and others, "Mobile-edge computing introductory technical white paper", *White Paper, Mobile-edge Computing (MEC) industry initiative*, 2014.
52. Hwang, Kai and Dongarra, Jack and Fox, Geoffrey C., "Distributed and cloud computing: from parallel processing to the internet of things", Morgan Kaufmann, 2013.
53. European-Commission, "Energy efficiency directive" https://ec.europa.eu/energy/en/topics/energy-efficiency/energy-efficiency-directive
54. Afman M., "Energiegebruik Nederlandse commerciele datacenters". http://www.cedelft.eu/publicatie/energy_consumption_of_dutch_commercial_datacentres%2C_2014-2017/1606
55. Huan L., "Host server CPU utilization in Amazon EC2 cloud", https://huanliu.wordpress.com/2012/02/17/host-server-cpu-utilization-in-amazon-ec2-cloud/
56. Altera Arria 10 FPGAs. https://www.altera.com/products/fpga/arria-series/arria-10/overview.html
57. Khronos Group, "The open standard for parallel programming of heterogeneous systems", https://www.khronos.org/opencl/
58. Thones J., "Microservices", *IEEE Software*, vol. 32, no. 1, 2015.
59. J. Stuecheli, B. Blaner, C. R. Johns and M. S. Siegel, "CAPI: A Coherent Accelerator Processor Interface" in *IBM Journal of Research and Development*, vol. 59, no. 1, 2015.
60. Organization for the Advancement of Structured Information Standards, "OASIS Topology and Orchestration Specification for Cloud Applications (TOSCA)", 2015.
61. Yaniv, Idan, and Dan Tsafrir, "Hash, Don't Cache (the Page Table)", *SIGMETRICS*, 2016.
62. C. Lefurgy, X. Wang, and M. Ware, "Server-level power control", *in Proc. of the IEEE International Conference on Autonomic Computing*, IEEE, 2007.