MEC-ConPaaS: An experimental single-board based mobile edge cloud

Alexandre Van Kempen, Teodor Crivat, Benjamin Trubert, Debaditya Roy, Guillaume Pierre

MEC-ConPaaS: An experimental single-board based mobile edge cloud

Alexandre van Kempen*, Teodor Crivat†, Benjamin Trubert‡, Debaditya Roy‡, and Guillaume Pierre‡

*INRIA, †Cloudtech Star Software, and ‡IRISA / University of Rennes 1

***Abstract*—Cloud infrastructures are extremely flexible and powerful, but their data centers are located very far from the end users. To address the limitations of these systems, many researchers are designing mobile edge clouds which complement traditional clouds with additional resources located in immediate proximity of the end users. However, currently there exists no open-source mobile edge cloud implementation which can easily be deployed over a campus or a city center to support real-world experimentations. We therefore present the design and implementation of MEC-ConPaaS, a mobile-edge cloud platform, which aims to support future research on edge cloud applications and middlewares. The system exploits single-board computers such as Raspberry Pis which are an order of magnitude cheaper than any server machine and much easier to setup in a distributed setting. We demonstrate that these devices are powerful enough to support real cloud applications, and to support further research on these topics.**

I. Introduction

Disruptive interactive end-user devices are reaching the consumer markets. Following the purchase by Microsoft of HoloLens (a headset which projects holographic images into real space [1]) and the $4.5 billion valuation of Magic Leap (a startup producing head-mounted displays for augmented reality applications [2]), similar technologies are now available in the consumer market with virtual reality headsets being commercialized for a few hundred Euros by HTC, LG, Oculus VR, Samsung, Sony, etc. Simultaneously, the very fast growth of Internet of Things brings a wide range of new products on the market from low-throughput sensor/actuator devices such as thermostats and connected toasters, to complex systems such as coordinated sets of surveillance cameras covering an entire neighborhood.

We can therefore expect the very fast emergence of a new family of *edge applications* which are defined as applications building a seamless interactive continuum between the physical and digital world, blending atoms and bits [3]. In the longer run, besides the obvious usage of these technologies for entertainment purposes, numerous applications are also expected in domains as diverse as live events coverage, healthcare, engineering, real estate, military, retail, and education [4].

Following the current trend of ever-increasing importance of mobile devices and infrastructures [5], end users will expect edge applications to be mobile. Applications will however require massive computation and storage to remain continuously available and responsive while processing potentially very large volumes of data. Yet, although large quantities of computing resources are readily available in cloud platforms, traditional cloud infrastructures are ill-equipped to fully address the challenges of future mobile edge applications. As they rely on a handful of very large data centers, public cloud providers are often located very far from their end users, which does not match the requirements of edge applications:

- **Interactive edge applications require ultra-low network latencies.** To guarantee an "instantaneous" feeling for the users, applications such as augmented reality require that end-to-end latencies (including all networking and processing delays) should not exceed 10-20 ms [6], [7]. However, measured latencies to the closest available data centers typically range from 20-30 ms using high-quality wired networks, up to 50-150 ms on 4G mobile networks [8], making traditional cloud resources unsuited for demanding applications.
- **Throughput-oriented edge applications require local computations.** The sources of application input data and the destination of computation results are often located geographically close to each other [9]. However, with only a handful of data centers from which to choose the location of data processing, input and output data are often transferred needlessly over long distances. This wastes long-distance networking resources, and may even create legal issues if the data center is not located in the same country as the end users.
- **Dependable edge applications must tolerate poor network connections.** Although cloud data centers are typically provisioned with excellent network connectivity, the same is not always true on the client side. Depending on the nature of the application, the availability of stable high-bandwidth wireless network connectivity may not always be guaranteed in locations relevant to the application. For example, an application dedicated to supporting emergency services during their operation must work seamlessly regardless of the location of the emergency (underground, in rural areas, etc.) where 4G network connectivity is often unreliable or even totally unavailable.

To address these challenges, the mobile networking industry is heavily investing in mobile edge cloud computing (MEC) platforms located at the edge of the networks, in immediate proximity to the end users [10], [11], [12]. Instead of treating the mobile operator's network as a high-latency dumb pipe between the end users and the external service providers, MEC platforms aim at deploying cloud functionalities *within* the mobile phone network, inside or close to the mobile access points. Doing so will deliver added value to the content providers and the end users by enabling new types of user experience. Simultaneously, it will generate extra revenue

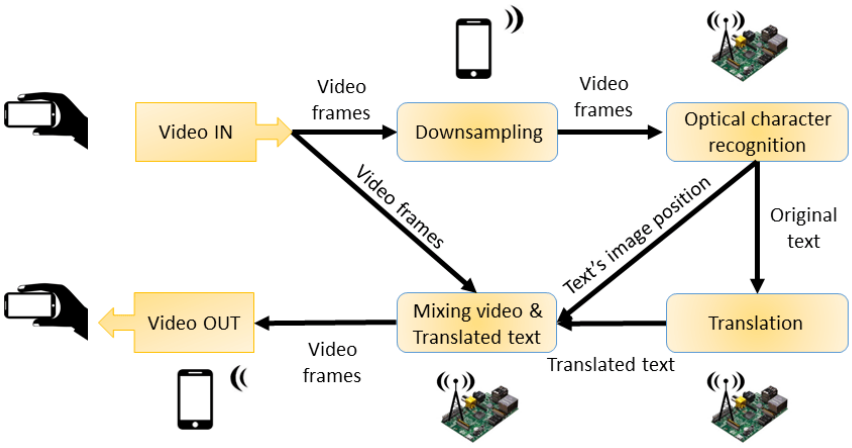

Fig. 1: Example of a stream-processing-based mobile edge cloud application

streams for the mobile network operators, by allowing them to position themselves as edge cloud operators and to rent their already-deployed infrastructure to content and application providers.

Mobile edge clouds have very different geographical distribution compared to traditional clouds. While traditional clouds are composed of many reliable and powerful machines located in a very small number of data centers and interconnected by very high-speed networks, mobile edge cloud are composed of a very large number of points-of-presence equipped with a couple of weak and potentially unreliable servers, and interconnected with each other by commodity long-distance networks. This creates new demands for the organization of a scalable mobile edge computing infrastructure, and opens new directions for research.

However, deploying a realistic mobile edge cloud remains a challenge. Mobile operators have not deployed MEC technologies in production yet, which makes it impossible for researchers to deploy their technologies in actual mobile phone networks. An alternative for them is to emulate a mobile edge cloud which uses Wi-Fi connectivity instead of LTE between the users and the edge cloud, but here as well there exists very few open-source platforms which may support such experimentation. The most mature one is probably OpenStack++ [13], but this system relies on classical server machines for its deployment. Deploying an experimental testbed reproducing a realistic geographical dispersion of computing resources therefore remains a challenge.

In this paper, we claim that an easier way to deploy an experimental MEC testbed is to rely on single-board computers such as Raspberry Pis (RPis) and similar devices. These devices are sufficiently inexpensive and power-efficient to be realistically deployed over a metropolitan area while providing sufficient computation power to support a large range of edge applications. We therefore discuss the design and implementation of an open-source platform for mobile edge clouds. For simplicity, we use Wi-Fi connectivity instead of LTE-based one where the same devices act both as a Wi-Fi hotspot and a cloud node, naturally providing extremely low network latency between end-user devices and the cloud instance(s) serving them.

The remainder of this paper is organized as follows. Section II discusses the state of the art. Then Sections III, IV, V and VI respectively discuss the system overview, hardware design, system-level and middleware design. We present the evaluation of an application deployed on Raspberry Pis in Section VII. Finally, Section VIII concludes.

II. State of the art

The trend towards pushing cloud computing to the edge of a network is expected to accelerate, and 2016 has already been named the "*year of the edge*" [14]. Edge computing originated from research efforts on Cloudlets, which propose to deploy a specific infrastructure at the edge of mobile networks to support dynamic application deployment in the immediate proximity of the end users [15], [16]. The outcome of this work is OpenStack++, an extension of the popular OpenStack cloud computing platform which includes mechanisms for application deployment in a fog computing context [13]. However, this platform is meant to execute on powerful server machines, which makes it difficult to deploy in a realistic geo-distributed setting.

From the point of view of interactive applications, it has been demonstrated that co-locating cloudlets with their end users drastically improves the execution of latency-critical applications [17]. However, the maximum tolerable distance between services and users depends on many factors including the level of interactivity of the application, the end-to-end network latency, the hosts hardware and software stack capabilities, and the users behavior. Similarly, the impact of user-service proximity in highly-interactive applications such as video streaming and collaborative chatting has been studied in [18]. A cloudlet-based approach always outperforms the classical cloud-based approach when clients are separated from their service with no more that two wireless network hops. Finally, the performance of different infrastructure settings (backend cloud, edge cloud and hybrid) are compared in [19]. They concluded that a smartly balanced hybrid infrastructure can bring the best of both worlds: the low latency of edge clouds and the high computing power of backend clouds.

As a result, many efforts focus on improving various aspects of mobile edge clouds. For example, the European

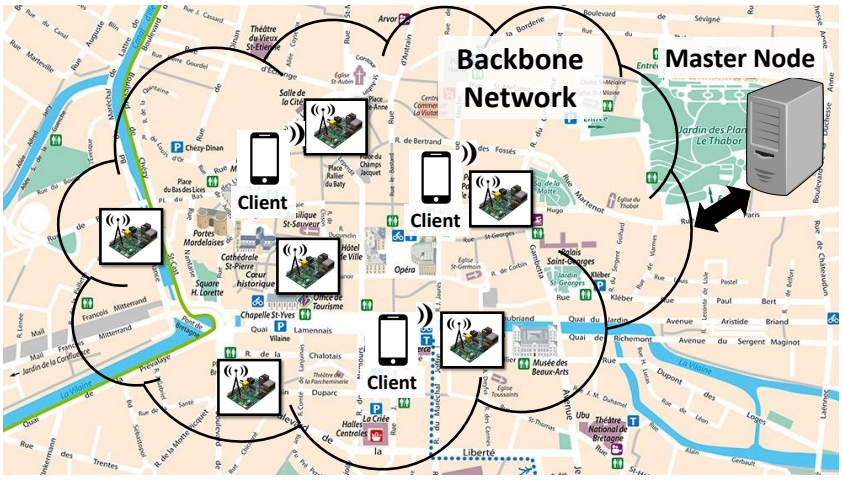

Fig. 2: Geographical distribution of a MEC-ConPaaS deployment.

Telecommunications Standards Institute's Mobile Edge Cloud group is busy designing ways to implement mobile edge clouds as a part of the upcoming 5G mobile network standards [10]. The OpenFog consortium [11] (ARM, Cisco, Dell, Intel, Microsoft, etc.) and the Open Edge Computing Initiative follow similar objectives with slightly different technical principles.

Many efforts also focus on the lower layers, down the the hardware level. For example, the recent ARM "Intelligent Flexible Cloud" (IFC) initiative pushes towards distributing intelligence towards the network edge [20]. In particular, a new family of processors, the ThunderX was specifically designed to support very efficient virtualization and contribute to Mobile Edge Computing platforms. ARM IFC also proposes an architecture that makes intensive use of emerging Network Function Virtualization (NFV) technologies.

Developers should be able to leverage the computing power of such mobile edge clouds, without having to handle the complexity of application deployment, fault tolerance issues, reconfiguration, or elasticity. Abstracting the complexity of the underlying platform enables developers to focus on application-specific concerns rather than on cloud-specific details. Platform-as-a-Service (PaaS) is a well known paradigm that provides services to developers, like commonly used runtime environments, so that existing applications can easily be ported to any cloud platforms.

In this paper, we rely on ConPaaS, an open-source runtime management system for elastic applications in public and private cloud environments [21]. We now extend this work to mobile edge cloud environments characterized by the small size and broad geographical distribution of its computing resources. Following on previous work which investigated virtualization technologies for single-board computers [22], [23], we rely on single-board computers such as Raspberry Pis and PINE A64+. However, contrary to these earlier contributions, we exploit these devices to build a widely-distributed mobile edge cloud rather than a single inexpensive data center. The closest work to ours has been recently published in [24], where the authors show the feasibility of using constrained nodes as fog gateways. However, this work target Internet-of-Things (IoT) scenarios where the fog gateways act as an intermediate layer between the IoT devices and a centralized cloud. In contrast, we seek to provide an open-source platform that enable developers to easily deploy applications within the fog platform itself, as close as possible to the end users.

III. System Overview

MEC-ConPaaS is a mobile edge cloud platform designed to be physically deployed across a city center or campus-sized geographical area, and to execute single-user and multi-user edge applications where proximity between the cloud instances and their end users is important. We first present a typical use case, then discuss the system architecture which supports it.

A. Use case

One application domain of mobile edge clouds is the tourism industry. When tourists visit a place unknown to them, they are in frequent need of local information and services to handle practical matters (local transportation, restaurants etc.), cultural ones (heritage, history, etc.) and entertainment. An example application that may be deployed for them is presented in Figure 1: this application captures a live video taken from the user's smartphone or tablet, identifies written signs from this video feed, and super-imposes a translated version of the recognized text in a different language.

This application is too complex to be executed in typical end-user mobile devices: it requires real-time optical character recognition, translation and video synthesis, which are beyond the processing capacity of a tablet or smartphone. Any application attempting to execute such application locally would quickly drain the battery. At the same time, there is no strong reason to upload the video feed to a cloud data center since both the source and destination of the workflow are located in the end user's device. Processing this workload as close as possible from the end user can only improve the application's responsiveness while saving precious long-distance networking resources.

Depending on the requirements of the application, specific components may be preferably located on the end-user device, the mobile edge cloud, or a traditional cloud backend. For example, the optical character recognition operator may require heavy real-time computations, and the translation operator may

require access to large volumes of data, two requirements incompatible with end users devices. These operators may therefore be placed in the mobile edge cloud, and dynamically scaled in and out according to the current workload. In addition, the appropriate network functions are automatically deployed and configured each time the requirements or the set of resources change, such that the application can abstract itself away from the architectural details of the underlying network.

B. Architecture

As shown in Figure 2, the MEC-ConPaaS platform can be deployed over single-board computers which are geographically distributed across the relevant area (such as a city center or a university campus). Each of the system nodes acts both as an access point and a cloud server: every node is equipped with a Wi-Fi interface to which client nodes can connect; at the same time, the same nodes operate as a virtualized infrastructure which can deploy applications in immediate proximity to the end users. The system is controlled by a traditional server which acts as the entry point of the system.

The system's architecture, depicted in Figure 3, comprises multiple layers that we describe hereafter from bottom to top. The hardware layer is composed of single-board computers such as Raspberry Pis, which provide the required computing resources, acting as light-weight cloud-like facilities. All these low cost devices are equipped with wifi interfaces to directly communicate with users at the edge. The interconnection between those devices can be ensured via a wired backbone network, or by leveraging their respective wifi interfaces. In order to provide elasticity and fault tolerance, applications are deployed inside a virtualized environment hosted on the same single-board devices. MEC-ConPaaS relies on LXC containers, which provide great levels of flexibility while being much lighter-weight than traditional virtual machines. These containers are orchestrated with OpenStack, while the ConPaaS layer, on top of OpenStack, eases the deployment of actual elastic applications [21]. Each of these layers is described more thoroughly in the next sections.

MEC-ConPaaS is readily available under a liberal open-source license[1]. It can be easily deployed over a set of single-board computers to deploy a mobile edge cloud platform on-the-fly, in geographical areas with limited coverage. MEC-ConPaaS may also be used to bring mobile edge applications during a specific event (e.g., a scientific conference or a rock concert). Finally, keeping an application's data flow inside a controlled and secured platform, instead of relying on external cloud operators, is another typical use case of MEC-ConPaaS.

IV. Hardware layer

MEC-ConPaaS relies on small single-board computers to act as both its Wi-Fi access points and its edge-cloud storage and computation servers. The rationale behind this choice is that simple-board computers offer a very interesting price/performance/power/volume tradeoff compared to traditional high-performance servers. For example, it is very easy to deploy a few dozen Raspberry Pi servers across a building. All that is required is to plug them in an electricity outlet and to connect them to the local LAN or Wi-Fi network. As

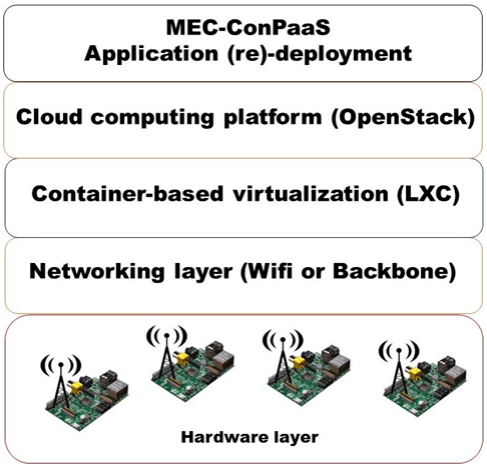

Fig. 3: MEC-ConPaaS' system architecture

weak as these devices might appear, we consider that they are largely powerful enough to constitute an excellent basis for a metropolitan edge cloud platform. To demonstrate the potential of these devices to act as cloud resources, in this section we measure and compare three single-board devices in terms of storage, CPU, memory, network and power consumption.

Raspberry Pis are the most famous line of single-board devices that may be used for such a purpose. However, other similar devices such as the PineA64+ or ODROID-C2 are currently reaching the market, each with their respective performance-price ratio. We compare the performance of two Raspberry Pi versions, the Raspberry Pi 2 and the Raspberry Pi 3[2] with the PineA64+ 2GB[3], and with a recent HP 820 G2 laptop which acts as a reference point. An overview of their respective hardware specifications is shown in Table I. Note that for the ARM boards (RPi2, RPi3, PineA64+), we tested different SD cards in order to ensure that the cards themselves were not the performance bottleneck. We report results based on class-10 Verbatim MicroSDHC cards.

The results of our benchmark measurements of these four devices are presented in Table II. We conducted every measurement 5 times. We removed the minimum and maximum values, and report the average value among the remaining three measurements. In all cases the standard deviations remained within 4% of the average.

Storage throughput: We compared the storage throughput using the *sysbench* utility, with a file size being the double of the available RAM to avoid caching effects. We report throughput results obtained under a random access pattern which is representative of the usage of a cloud server. The 32 GB card we tested is clearly the bottleneck here, as the three boards all provide a throughput around 2.5 MB/s. In comparison, the laptop's throughput based on a state-of-the-art SSD drive is an order of magnitude faster.

CPU performance: The CPU score is obtained via the *sysbench* utility as well, using 4 threads in order to fully utilize the quad-core CPUs. Interestingly, the PineA64+'s performance comes close to that of the laptop, while there is a

[1] http://www.conpaas.eu

[2] https://www.raspberrypi.org/

[3] https://www.pine64.org/

TABLE I: Hardware specifications.

	RPi2	RPi3	Pine A64+	HP 820 G2
Machine type	Single-board	Single-board	Single-board	Laptop
Storage	32 GB microSD card	32 GB microSD card	32 GB microSD card	512 GB SSD drive
CPU	Broadcom Quad-core 900MHz 32 Bits ARM Cortex-A7	Broadcom Quad-core 1.2GHz 64 Bits ARM Cortex-A53	AllWiner Quad-core 1.2GHz 64 Bits ARM Cortex A553	Intel 2 cores – 4 Threads 2.6GHz 64 Bits Intel core i7 5600U
Memory	1GB 450MHz LPDDR2	1GB 900MHz LPDDR2	2GB DDR3	16GB 1600MHz DDR3L
Network	100 Mbps	100 Mbps	1Gbps	1Gbps
Price incl. storage	~ 82 €	~ 92 €	~ 74 €	~ 1600 €

TABLE II: Performance comparison between 3 ARM-based single-board computers and a reference laptop.

	RPi2	RPi3	Pine A64+	HP 820 G2
Storage (MB/s)	2.45	2.53	2.42	23.9
CPU (s)	82	46.5	4.1	2.8
Memory (MB/s)	272	933	1098	5658
Network (Mb/s)	94.1	94.2	922	935
Power when idle (W)	2	2	2	15
Power under load (W)	3.6	4.4	4.1	24.5

significant difference between the PineA64+ and the Raspberry PIs. This comes from the fact that *sysbench* has been optimized for 64 bits instructions and the Raspberry Pis are still running a 32-bits operating system. We therefore expect that future single-board devices will feature CPU performance that is at least on-par with the PineA64+.

Memory throughput: we measured the memory throughput using the *mbw* tool. Results show little difference between the Raspberry Pi 3 and the PineA64+, which both outperform the Raspberry Pi 2.

Network throughput: measurements were conducted using *iperf* and are in compliance with each device networks cards. Interestingly, the PineA64+ is capable of saturating a gigabit link. In comparison the Raspberry Pis' network operates over the USB bus, which strongly limits the network throughput. Here as well we can expect that future versions of the Raspberry Pi may feature significantly greater network capacity.

Power consumption: we measured the devices' power consumption using a power meter placed between the device and the electric outlet. To avoid any interference with the laptop's battery, we removed the battery during these measurements. We respectively measured the electrical consumption when the devices were idle and under full CPU load for two minutes, with one measurement taken every 5 seconds. When idle, the laptop and all the three boards were respectively consuming around 15 Watt and 2 Watt, on average. When running at full load, the Raspberry Pi 2, Raspberry Pi 3 and the PineA64+ were respectively consuming 3.6, 4.4 and 4.1 W in average, while the laptop reached 24.5 W.

Lastly, as explained in the previous section, MEC-ConPaaS heavily relies on containers to provide elasticity. We thus also ran every test inside an LXC container to evaluate the overhead due to virtualization. In all our tests, the virtualization overhead remained below 6%.

The conclusion of these benchmarks is that the single-board devices offer a very interesting performance/price tradeoff compared to the tested laptop. In particular, the Pine A64+ performs very close to the laptop regarding the CPU, memory and networking components, while being over one order of magnitude cheaper. We believe it demonstrates the ability of single-board devices to deliver excellent performance/price ratios. The only component for which it is still significantly lagging behind is storage, which is inherently limited by poor performance of the SD-card technology. However, newer devices such as the ODROID C2 are now starting to use (hopefully faster) eMMC storage devices.

The Raspberry Pis represent a slightly older generation of single-board devices. However, we expect that future versions are going to keep up with these very fast hardware developments. On the other hand, the Raspberry Pis excel in other dimensions such as the size and reactiveness of their user communities, and the variety of retail options. For these reasons, we based the implementation of MEC-ConPaaS (and the evaluation section of this paper) on Raspberry Pi 3 devices, in anticipation of future hardware improvements of Raspberry Pi devices.

V. System layer

Although single-board computers such as Raspberry Pi and PineA64+ are very capable machines, using them as compute and storage nodes in a cloud setup is not an easy exercise. The challenges are mainly twofold: (i) updating the provided operating system and libraries so that they integrate the relevant support for virtualization; and (ii) addressing the performance limitations of the single-board devices to use them at the best of their capacity.

A. Virtualization support

The recent ARM processors which power the vast majority of single-board computers include hardware support for virtualization. However, we decided to base ourselves on LXC containers rather than virtual machines. The main reason is memory: single-board computers have very limited main memory size (between 1 and 2 GB), which strongly limits the number of VMs which could be executed simultaneously on a single device. In contrast, containers are much more lightweight in this respect, which proved to be essential for resource-constraint devices such as Raspberry Pis.

This however implies that the Linux kernel must be compiled with LXC-specific options enabled, like *cgroups* or *cpuset* support, and a *libvirt* API with full support of LXC. Note that depending on the board used, some recent distributions such as Debian 8 now natively enable those options, contrary to what was available at the time of our

experiments. In addition, one must configure OpenStack to make use of the LXC driver installed on every compute nodes. Note that we released MEC-ConPaaS in the form of ready-made images [4] that can be directly copied on the intended devices so no specific configuration is required from the MEC-ConPaaS users.

B. Performance tuning

Even though single-board computers are becoming increasingly powerful, they still have many performance limitations. In our experience, the most challenging bottleneck that must be addressed to obtain decent performance is the very slow I/O throughput highlighted in Section IV, together with the limited main memory size. It is therefore essential to carefully customize the default operating system configuration to avoid common pitfalls. The following optimizations enabled us to reduce the duration of a container creation operation from 10 minutes to about 90 seconds on a Raspberry Pi 2.

The first obvious optimization is to strip the Raspberry Pi devices from all the unnecessary software and services. OpenStack services can be very memory-hungry so we installed only the strict minimum on the Raspberry Pi devices: *nova-compute* (which controls the creation/deletion of LXC containers) and *cinder-volume* (which controls the creation/attachment/detachment/deletion of storage volumes). All other OpenStack services related to authentication, virtual disk image management and such, are rather installed on a single powerful "master node" which acts as the entry point of the system.

The SD cards which act as the only storage device of all our single-board computers are not meant to store a Linux file system composed of a myriad of small files: SD cards are mostly used in mobile phones and similar devices to store a relatively small number of large files such as photos and videos. As a result, these cards are designed to use relatively large block sizes, in the order of 4 MB. It is therefore essential to align the file system partitions on the SD card's erase block boundaries to avoid generating unnecessary I/O upon any file system operation.

Finally, an important kernel parameter for the overall system performance is the so-called *swappiness*. This parameter controls the relative weight given to swapping out runtime memory, as opposed to dropping pages from the system page cache. A low value causes the kernel to avoid swapping as much as possible while a higher value causes the kernel to use the swap space much more proactively. In most Linux distributions for single-board computers, SD-card throughput is known to be an important performance bottleneck and, hence, the default swappiness value is set to 1 out of the $[0; 100]$ interval of possible values. This means that the system should swap only when absolutely necessary. For most purposes where the available memory is sufficient to hold the entire runtime memory, this is indeed a valid configuration.

However, executing a few OpenStack daemons and starting a new container consumes a lot of memory, and often temporarily exceeds the available memory on a Raspberry Pi. With a very low swappiness value, the system can page memory only at the time a new container is being created, which considerably slows down this creation.

Another indirect consequence of a low swappiness value is that it leaves little space (if any) to store the file system cache: in Linux, the file system cache occupies the currently unused main memory. Without a disk cache, every single file access is systematically translated in one or more I/O operations addressed to the SD card. A container creation operation accesses many files such as the usual system binaries, the Python runtime, configuration files and such, which therefore produces a sudden burst of I/O operations which considerably delay the container creation.

In our use case, many of the running services are used only for a brief moment during the container creation operation. These memory pages should thus be swapped out as soon as possible, in order to make more room for actual application memory and for the disk cache. We therefore set the swappiness to the greatest possible value (100), which offered major performance improvements. Note that this configuration does not increase the total volume of data which has to be written/read to/from the SD card: in fact, it actually *reduces* the total number of I/O operations compared to the original swappiness value.

VI. Middleware layer

Deploying a complex application on a traditional cloud platform is not an easy task; deploying it on an edge cloud will certainly be even less so, as one has to deal with extra concerns such as the geographical location of the end users and the cloud resources supporting their applications. In order to ease this deployment, we leverage the ConPaaS system, which is an open-source multi-cloud run-time management system for elastic applications [21]. ConPaaS lets its users upload simple application code, and takes care of handling the complexity of deploying such applications in cloud environments. It automates the entire life-cycle of an application, including collaborative development, deployment, performance monitoring, and automatic scaling.

In ConPaaS, applications are organized as a collection of services. ConPaaS currently provides ready-made services for a number of commonly-used middlewares: web hosting (PHP and Java), SQL and NoSQL databases (MySQL and Scalaris), data storage (XtreemFS) and for large scale data processing (Task Farming, MapReduce and Apache Flink). Applications making use of these middlewares only need to create the appropriate service and upload the relevant code or data to this service. The system automatically takes care of provisioning one or more containers, deploying the application, implementing horizontal elasticity, etc.

Applications which do not fit in this model may make use of the "Generic" service, which enables users to package arbitrary code and data, and deploy it similarly to the other services. The only difference is that application-specific operations such as the dynamic addition and removal of computing resources must be implemented by the application rather than by the ConPaaS system.

The horizontal elasticity properties of ConPaaS are also useful to implement seamless service migration. When a user

[4] http://www.conpaas.eu/download/

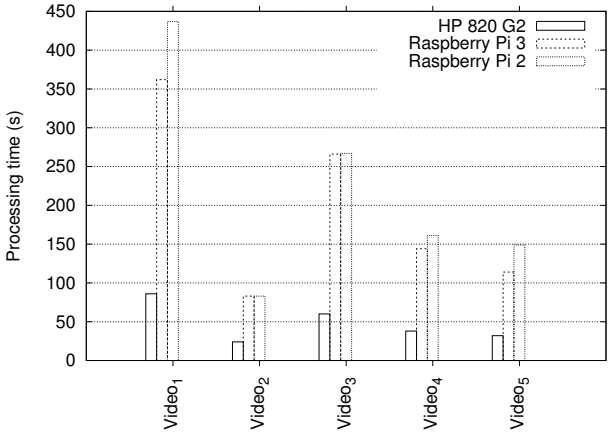

Fig. 4: Comparison of processing times depending on the different computing units

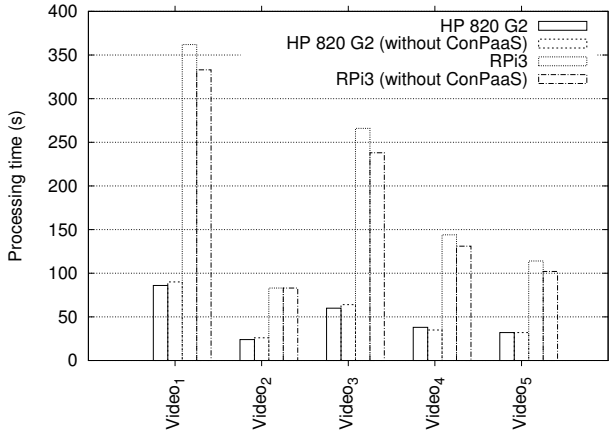

Fig. 5: ConPaaS overhead evaluation

moves around the area covered by the mobile edge cloud, it may be desirable to migrate the applications serving her such that the application is constantly within very short network distance to the user. If the applications are elastically scalable, this migration may be elegantly implemented by creating a new instance of the application in the desired destination node, followed by removing the original instance in the previous location.

Finally, ConPaaS was originally designed to operate over a variety of public and private clouds such as Amazon EC2, OpenStack and OpenNebula. We extend, in this paper, the underlying cloud infrastructures to include mobile edge cloud environments that may help an application to place its points of presence close to its users, thus performing computations more locally and reducing the latencies.

VII. Evaluation

We now illustrate the way an application developer can exploit MEC-ConPaaS to create innovative applications supported by mobile edge cloud technologies. To this end, we make use of a simple example application which performs face recognition in a live video stream [25].

The application is implemented using the Apache Flink stream-processing middleware [26]. Although the stream processing model was initially designed to handle data analytics applications, we believe it is particularly well-suited for developing mobile edge applications as it provides developers with an easy-to-understand development environment, while being able to harness the full capacity of cloud infrastructures to achieve extremely high performance. In addition, the organization of the application as a workflow of operators offers a simple yet powerful abstraction which facilitates the deployment and run-time management of the application in complex multi-cloud environments.

Several mature stream processing platforms are currently available in open-source, the most notable ones being Apache Storm [27], Apache Spark [28] and Apache Flink [29]. In this example we chose to rely on the Flink platform as it combines benefits inspired by Storm and Spark: like Storm, Flink provides low-latency stream processing by pipelined data transfers that can deliver high throughput across a wide spectrum of latencies and strong consistency guarantees even in the presence of stateful computations. Like Spark, Flink offers a high-level API. Also, unlike Apache Spark, which breaks down continuous computation into a series of small, atomic batch jobs (micro-batching), Flink is primarily a stream processing framework, which makes it particularly suitable for hyper-interactive applications where low latency is a requirement.

With this example, we show how MEC-ConPaaS offers a new application domain for stream-processing systems in a fog context, as it automatically deploys, migrates and reconfigures applications to maintain their performance close to optimal despite arrivals or departures of users into/from the application, variations in the workload imposed by the users, or end users mobility. This will benefit both application developers who will be able to exploit the full power of fog computing while retaining an intuitive programming model, and the stream-processing community which will expand their scope to new customers and application scenarios.

The face recognition application is a small Flink program, written in Java, which leverages the OpenCV library[5] to perform image processing. This library unfortunately does not implement any form of parallelism, thus limiting the execution of the face recognition functionality in a single thread. Thanks to the Apache Flink service provided by ConPaaS, a user can simply upload this application to the MEC-ConPaaS system which then automatically deploys the application on the underlying cloud infrastructure.

Figure 4 show the resulting processing times for 5 different videos, and for three computing platforms: the laptop configuration and the two Raspberry Pi versions. Note that we did not include the Pine A64+ in the evaluation, as ConPaaS was not yet fully ported on this platform at the time of writing. These results show that, even though a Raspberry Pi cannot keep up with the performance of a recent laptop, the ratio of execution times between the laptop and raspberries is only 3 or 4. This is remarkable given that the microbenchmarks presented in section IV rather showed an order of magnitude between the two device families. In addition, the Raspberry Pi 3 offers performance improvements up to 30% on some

[5] http://opencv.org/

videos when compared to the Raspberry Pi 2, as expected in regards with the microbenchmarks section. We unfortunately could not test the horizontal scalability of the system because the compute-intensive part of the application relies on the single-threaded OpenCV library. However, an improved multithreaded version of the same application should be able to exploit the capabilities of multiple co-located compute node, and to reach comparable levels of performance to that of a high-performance server[6].

Finally, we evaluated the overhead due to the ConPaaS deployment system on the processing time by comparing performance with that of a traditional deployment with no virtualization nor cloud technologies. We clearly see in Figure 5, that ConPaaS generates a very small overhead both on the Raspberry Pi 3 or the laptop configuration, whereas it considerably eases the application deployment on the computing units.

VIII. Conclusion

Cloud infrastructures are extremely flexible and powerful, but their data centers are located very far from the end users. This creates a need for complementing these infrastructures with additional "mobile edge cloud" resources located at the edge of the network, in immediate proximity of the end users. However, deploying a realistic mobile edge cloud remains a challenge, which unnecessarily limits the possibilities for researchers to experiment with various types of mobile edge applications and middlewares.

We presented MEC-ConPaaS, an open-source platform which aims to support further research efforts in the domain of mobile edge cloud applications and middleware. MEC-ConPaaS is designed to be extremely easy to deploy over a university campus or a city center, and to operate using lightweight and inexpensive single-board computers.

We demonstrated that single-board computers, if exploited correctly, have enough capacity to process serious mobile edge applications. We based our implementation on Raspberry Pi devices, which have the advantage of being very well supported and easy to purchase. The very fast development of other brands of single-board devices suggest that future generations of single-board computers will deliver even much greater performance, while retaining the same very low cost, volume, and power consumption as the devices available to us today.

References

[1] Microsoft Corporation, "Microsoft HoloLens," http://www.microsoft.com/microsoft-hololens.

[2] I. Lunden, "AR Startup Magic Leap Raises $793.5M Series C At $4.5B Valuation Led By Alibaba," Tech Crunch, Feb. 2016, http://bit.do/bS4iU.

[3] R. Saracco, "The fading line between atoms and bits," *IEEE Internet of Things Newsletter*, Sep. 2014.

[4] Goldman Sachs, "Virtual & augmented reality: The next big computing platform?" Equity research, Feb. 2016, http://bit.do/bS4jc.

[5] GSM Association, "The mobile economy," 2016, http://www.gsmamobileeconomy.com/.

[6] M. Abrash, "Latency – the sine qua non of AR and VR," Blog post, Dec. 2012, http://bit.ly/1ERjiOw.

[7] OculusRift Blog, "John Carmack's delivers some home truths on latency," http://bit.do/bS4jg.

[8] CLAudit Project, "Planetary-scale cloud latency auditing platform," 2016, http://bit.do/bS4js.

[9] S. Scellato, C. Mascolo, M. Musolesi, and V. Latora, "Distance matters: Geo-social metrics for online social networks," in *Proc. WOSN*, Jun. 2010.

[10] European Telecommunications Standards Institute (ETSI), "Mobile edge cloud," Industry Specification Group (ISG), 2014, http://bit.do/bS4jL.

[11] OpenFog Consortium, 2016, http://www.openfogconsortium.org.

[12] Open Edge Computing Initiative, 2016, www.openedgecomputing.org.

[13] K. Ha and M. Satyanarayanan, "OpenStack++ for cloudlet deployment," Carnegie Mellon University, Research Report CMU-CS-15-123, Aug. 2015.

[14] EdgeConneX, "Edgeconnex® announces findings from webinar on improved internet performance via edge infrastructure deployments," Press release, Jan. 2016, http://bit.do/bTzxb.

[15] M. Satyanarayanan, P. Bahl, R. Caceres, and N. Davies, "The case for VM-based cloudlets in mobile computing," *IEEE Pervasive Computing*, vol. 8, no. 4, 2009.

[16] G. A. Lewis, S. Echeverría, S. Simanta, B. Bradshaw, and J. Root, "Cloudlet-based cyber-foraging for mobile systems in resource-constrained edge environments," in *Proc. ICSE*, 2014.

[17] S. Clinch, J. Harkes, A. Friday, N. Davies, and M. Satyanarayanan, "How close is close enough? Understanding the role of cloudlets in supporting display appropriation by mobile users," in *Proc. PerCom*, 2012.

[18] D. Fesehaye, Y. Gao, K. Nahrstedt, and G. Wang, "Impact of cloudlets on interactive mobile cloud applications," in *Proc. EDOC*, 2012.

[19] M. Baguena, A. Pamboris, P. Pietzuch, G. Samaras, M. Sichitiu, and P. Manzoni, "Towards enabling hyper-responsive mobile apps at scale using edge assistance," in *Proc. CCNC*, Jan. 2016.

[20] J. Fry, "The intelligent flexible cloud," ARM® white paper, Feb. 2015, http://bit.do/bS4j6.

[21] G. Pierre and C. Stratan, "ConPaaS: a platform for hosting elastic cloud applications," *IEEE Internet Computing*, vol. 16, no. 5, 2012.

[22] F. P. Tso, D. R. White, S. Jouet, J. Singer, and D. P. Pezaros, "The glasgow raspberry pi cloud: A scale model for cloud computing infrastructures." in *ICDCS Workshops*, 2013.

[23] P. Abrahamsson, S. Helmer, N. Phaphoom, L. Nicolodi, N. Preda, L. Miori, M. Angriman, J. Rikkilä, X. Wang, K. Hamily, and S. Bugoloni, "Affordable and energy-efficient cloud computing clusters: The bolzano raspberry pi cloud cluster experiment," in *CLOUDCOM '13*, 2013.

[24] P. Bellavista and A. Zanni, "Feasibility of fog computing deployment based on docker containerization over raspberrypi," in *Proc. ICDCN*, 2017.

[25] A. Tetzlaff, A. Getzin, T. Tran, and J. Kirschnik, "Finding faces," TU Berlin database project, https://github.com/jkirsch/senser.

[26] A. Alexandrov, R. Bergmann, S. Ewen, J.-C. Freytag, F. Hueske, A. Heise, O. Kao, M. Leich, U. Leser, V. Markl, F. Naumann, M. Peters, A. Rheinländer, M. J. Sax, S. Schelter, M. Höger, K. Tzoumas, and D. Warneke, "The Stratosphere platform for big data analytics," *VLDB Journal*, vol. 23, no. 6, Dec. 2014.

[27] Apache Software Foundation, "Apache Storm," https://storm.apache.org/.

[28] M. Zaharia, M. Chowdhury, T. Das, A. Dave, J. Ma, M. McCauley, M. J. Franklin, S. Shenker, and I. Stoica, "Resilient distributed datasets: A fault-tolerant abstraction for in-memory cluster computing," in *Proc. NSDI*, Apr. 2012.

[29] Apache Foundation, "Apache Flink project," 2016, https://flink.apache.org/.

[6] Single-board computers are also often equipped with one or more GPUs which could also be used to speedup this type of computations.