Empowering Low-Latency Applications Through a Serverless Edge Computing Architecture

Luciano Baresi, Danilo Filgueira Mendonça, Martin Garriga

Empowering Low-latency Applications through a Serverless Edge Computing Architecture

Luciano Baresi, Danilo Filgueira Mendonça, and Martin Garriga

Dipartimento di Elettronica, Informazione e Bioingegneria,
Politecnico di Milano, Italy
{luciano.baresi,danilo.filgueira,martin.garriga}@polimi.it

Abstract. The exponential increase of the data generated by pervasive and mobile devices requires disrupting approaches for the realization of emerging mobile and IoT applications. Although cloud computing provides virtually unlimited computational resources, low-latency applications cannot afford the high latencies introduced by sending and retrieving data from/to the cloud. In this scenario, edge computing appears as a promising solution by bringing computation and data near to users and devices. However, the resource-finite nature of edge servers constrains the possibility of deploying full applications on them. To cope with these problems, we propose a serverless architecture at the edge, bringing a highly scalable, intelligent and cost-effective use of edge infrastructure's resources with minimal configuration and operation efforts. The feasibility of our approach is shown through an augmented reality use case for mobile devices, in which we offload computation and data intensive tasks from the devices to serverless functions at the edge, outperforming the cloud alternative up to 80% in terms of throughput and latency.

Keywords: serverless architectures, edge computing, mobile edge computing, low-latency applications

1 Introduction

Mobile data will skyrocket in the coming years, mainly driven by mobile video streaming and the Internet of Things (IoT). In 2017, data traffic of mobile devices is expected to exceed 6 Exabytes ($6 * 10^9$ Gigabytes) per month, and when combined with the traffic generated by laptops and machine-to-machine communications, the overall demand should reach 11 Exabytes per month [1]. Although cloud computing appears as a straightforward solution for processing such an amount of data, in certain scenarios the latency introduced by sending/retrieving heavy payloads from/to the cloud can be prohibitive [2]. To address data-intensive and low latency requirements, as well as to avoid the bottlenecks of centralized servers, edge computing proposes to bring computation to the edge of the network, that is, near to where it is needed by users and devices [3]. Moreover, Mobile Edge Computing (MEC) allows for the use of its services with low latency, location awareness and mobility support to make up for the disadvantages of cloud computing [4].

However, the distributed and resource-finite nature of edge infrastructure also imposes limitations regarding its capability of hosting many diverse applications and/or services, otherwise hosted remotely in the cloud [3], since an overloaded MEC server significantly degrades user experience and negates the advantages of MEC [4]. Thus, such an scenario cannot be simply supported by a straightforward migration of the existing cloud model at the edge, that is, simply adopting *virtualization* and *containerization* technologies [5]. Recently, *Serverless Architectures* [6], also known as *Functions-as-a-Service (FaaS)*, appeared as a disruptive alternative that delegates the management of the execution environment of an application (in the form of stateless functions) to the infrastructure provider [7]. As a consequence, provider-managed containers are used to execute functions, without pre-allocating any computing capability or dealing with scalability and load-balancing burden. This should boost the utility of the edge nodes, allowing one to deploy more functionality given their limited capabilities and resources, while meeting application's low latency requirements.

This paper presents a *serverless edge computing* architecture that enables the offloading of mobile computation with low latency and high throughput. The objective is to allow low-latency mobile applications to minimize the impact on the resources of devices (which are battery and CPU constrained) and satisfy their latency requirement. The feasibility of the proposed architecture is evaluated through a mobile augmented reality application, and compared against a cloud-based solution. Results show that, in data-intensive scenarios, the proposed serverless edge solution outperformed the cloud-based offloading solution up to 80% in terms of throughput and latency.

The rest of the paper is organized as follows. Section 2 defines edge computing and serverless architectures. Section 3 presents a motivating case study: a Mobile Augmented Reality application. Section 4 describes the proposed architecture. Section 5 presents the evaluation we carried out. Section 6 discusses related work. Section 7 concludes the paper.

2 Background

Edge computing is a distributed computing paradigm that aims to cope with the rapid increase in data coming from the plethora of mobile devices. Its main purpose is to boost the potential of the Internet-of-Things and other real-time and data-intensive applications [3, 8], by shifting the computation from the center (server) of the system towards a computing infrastructure deployed at the edges of the system (or of the network). The aim is to mitigate the latency and bottlenecks of centralized or coarsely distributed servers.

In contrast to the more general term, Mobile Edge Computing (MEC) focuses on co-locating computing and storage resources at base stations of cellular networks, thus reducing the stress of the network by shifting computational efforts from servers deployed in the Internet to the edges of the mobile network [3, 9]. Being co-located at base stations, computing and storage resources of MEC servers are also available in close proximity to mobile users, thus eliminating

the need for routing these data through the core network. MEC is seen as a future and promising approach to increase the quality of experience in cellular networks, and a key enabler for the evolution to 5G networks [10]. A distributed PaaS (Platform as a Service) can be deployed within the radio access network to serve low-latency, context-aware applications timely.

A Serverless Architecture is a refined cloud computing model to process requested functionality without pre-allocating any computing capability. Provider-managed containers are used to execute functions (often called lambdas), which are event-triggered and ephemeral (may only last for one invocation) [6]. This approach allows one to write and deploy code without considering the runtime environment, resource allocation, load balancing, and scalability; all these aspects are handled by the provider.

The serverless model represents a further evolution of the pay-per-use computing model: we started allocating virtual machines (e.g., Amazon EC2), then moved to containers (e.g., CS Docker Engine) and now we only allocate the resources (a container shared by several functions) for the time needed to carry out the computation.

The Serverless architecture has many benefits with respect to more traditional, server-based approaches. Functions share the runtime environment (typically a pool of containers), and the code specific to a particular application is small and stateless by design. Hence, the deployment of a pool of shared containers (workers) on a machine (or a cluster of machines) and the execution of some code onto any of them becomes inexpensive and efficient.

Horizontal scaling is completely automatic, elastic, and quick, allowing one to increase the number of workers against sudden spikes of traffic. The serverless model is much more reactive than the typical solutions of scaling virtual machines or spinning up containers against bursts in the workload [11]. Finally, the pay-per-use cost model is fine-grained, down to a 100ms granularity for all the major vendors, in contrast to the "usual" hour-based billing of virtual machines and containers. This allows companies to drastically reduce the cost of their infrastructures with regard to a typical monolithic architecture or even a microservices architecture [12].

Several cloud providers have developed serverless solutions recently, many of which are still in their explicit or implicit beta testing phase[1]. Table 1 summarizes the main serverless solutions, with AWS Lambda that appeared 1.5 years before the others. All these alternatives provide similar capabilities; IBM Openwhisk is the only open-source solution among the major vendors.

3 Mobile Augmented Reality

Augmented reality (AR) is the combination of a view of the real world and supplementary computer-generated information [10]. More recently, Mobile Augmented Reality (MAR) emerged as a fusion of AR and mobile computing. MAR

[1] https://blog.zhaw.ch/icclab/faas-function-hosting-services-and-their-technical-characteristics

Table 1. Serverless providers and supported languages

Provider	Languages
AWS Lambda	Node.js, Java, Python
Google Cloud Functions	Node.js
Azure Functions	Node.js, C#
IBM OpenWhisk	Node.js, Swift, Binary (Docker)
Webtask.io	Node.js
OpenLambda	Python

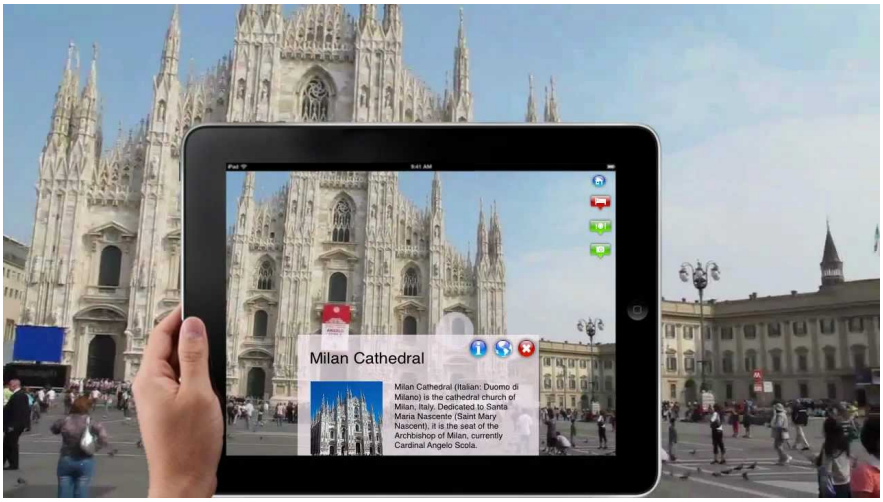


Fig. 1. An example Mobile Augmented Reality app (etips.com).

is an example of applications for which low latency and high throughput are key requirements. These applications enrich the interaction of users with the physical world by augmenting their vision of the reality with relevant information (e.g., historical information about buildings and monuments), modifying it (e.g., by translating captured text in a different language), or by adding virtual elements that can mimic interactions with the real world (e.g., virtual objects or creatures from a fantasy game), or helping users fulfill physical tasks (e.g., by highlighting a free parking spot).

Our example MAR application is supposed to help the tourists that visit a city and want to receive relevant information about Points-of-Interest (POIs), such as monuments, buildings, and other architectural elements, by looking at them through their mobile devices (Figure 3) or special glasses [13].

Based on the approach described by Huang et al. [14], the following steps summarize the sequence of data- and computational-intensive tasks in MAR applications:

1. The reality that must be augmented should be captured by using the device's camera, with a rate between 2 and 6 images per second [15, 16].
2. The captured frame must be scanned to extract the features that allow the app to identify the physical objects in the scene.
3. Virtual content, associated with the identified scene and objects, must be retrieved from servers[2] based on the previously extracted features.
4. Finally, the app produces a combined image of the real and virtual contents and displays it on the device screen.

As users can rapidly move and target different portions of the world around them, target scenes must be captured by the device's camera at a fast rate (step 1), generating a significant volume of data frequently. Also, the extraction of features from the objects in these frames (step 2) is a computational-intensive task. Prohibitive network traffic and latency can be avoided by letting step 2 be performed locally and delegating only steps 3 and 4 to services in the cloud [14]. However, this kind of approach may fail to meet users' expectations because continuously transferring information to cloud services and interacting with them could be slow, and it can significantly reduce the battery of their devices [17]. Offloading mobile computation to a MEC platform rather than using "traditional" cloud services should bring several advantages: First, it provides the low latency and high throughput required by mobile augmented reality applications; second, it prevents the overloading of mobile devices with computational-intensive tasks; and finally, the MEC platform can adjust provisioned resources on-the-fly and no resources are wasted.

4 Proposed Solution

Figure 2 shows the proposed architecture. Its main physical elements are mobile devices and MEC servers. Mobile devices can be of any type (e.g., tablets, smartphones), running a low-latency application that needs offloading part of its computation to more powerful servers. For this, the devices send the information to be processed to the MEC server through standardized network protocols [18]. A Base Transceiver Station[3] (BTS) bridges mobile devices and MEC servers as a part of the cellular infrastructure and MEC architecture, according to its current specifications [10]. In this scenario, mobile devices and MEC servers are at no more than a few hops from each other. MEC servers host the serverless environment, where stateless functions are deployed and executed.

While MEC servers are ideal candidates for offloading the computation to preserve devices' resources and kill latency, these nodes are themselves potentially constrained. Accordingly, the feasibility of hosting dedicated *virtual machines*, *containers*, and *stateful applications* would also be limited, as these nodes

[2] This information cannot be usually stored on the device given its size and dynamic nature.

[3] Different generations of wireless mobile networks use distinct names (e.g., eNodeB in 4G).

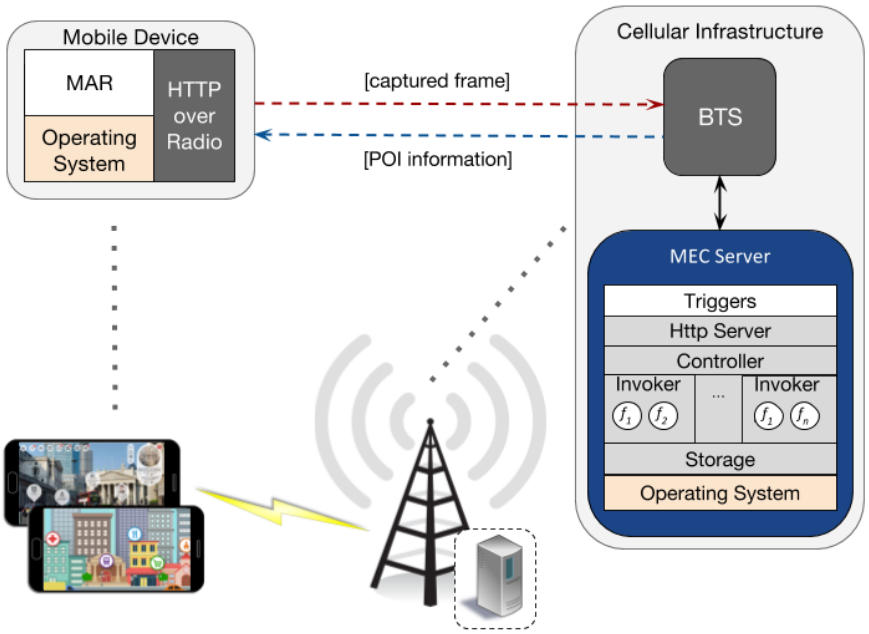


Fig. 2. Proposed Architecture: A MAR application running on mobile devices send requests to the MEC server hosted on a cellular infrastructure (shared components of the serverless MEC server are depicted in grey).

cannot scale "infinitely" to host always-running VMs/containers as the cloud itself. To overcome this limitation, we propose to deploy a serverless architecture [6] onto the MEC servers.

Figure 2 also shows the serverless components deployed on the MEC server. The entry points are the *triggers* associated with events: in the MAR application, an event that triggers a function consists of uploading of an image or capturing a frame with the device's camera. These triggers fire requests to an *Http Server* that exposes a Restful API of available functions.

To achieve network transparency, a local Domain Name Server (DNS), deployed on the cellular infrastructure, must distinguish between requests to the RESTful APIs exposed by the MEC server and any other request for an Internet endpoint. The main difference from a regular DNS is locality, as the requests must be handled by the MEC server on the current base station. To this end, the names of edge resources must be resolved locally without being propagated to public DNS servers. Whereas the specific details of the naming solution are outside the scope of this work, we argue that such a feature should not pose a significant technical challenge.

Once a request reaches the MEC server, it is then forwarded to a *controller* component, which identifies and retrieves the function being called, authorizes the execution of such a function and identifies an available invoker to run it. *In-*

vokers isolate the *functions* in containerized environments, optimized and managed by the serverless provider to reduce overhead and response time. Finally, results and logging information are stored in the *Storage* component, a highly available, noSQL database.

Note that most of the components of the serverless architecture of the MEC server are shared (in grey in Figure 2) among all the functions. The highly shared nature and the automated management of the whole platform allows any function deployed on the MEC servers to scale up automatically and elastically to unexpected bursts in the workload, and to scale down when it is not used anymore. In contrast with container-based stateful applications, the serverless platform is responsible for allocating functions of one or more applications on a pool of containers according to the resources available at the MEC server. As a result, the use of the computational resources of MEC servers is optimized, allowing both more functions to be deployed and more requests to be processed simultaneously. A conventional cloud provider can always become part of the deployment if needed, but it is not the focus of this paper.

There is no need to follow the common practice of deploying multiple virtual machines or containers to be resilient and responsive against downtime of single instances or bursts of workload. The on-demand execution of functions provides inherent scalability and optimal utilization as the number of running functions always matches the trigger rate. Additionally, the application developer only focuses on the application code and can fully outsource the management of the deployment/execution infrastructure. The serverless approach also provides a fine-grained *pay-per-use* billing model with benefits for both application owners and telecom operators (in charge of the MEC servers).

4.1 Mobile Augmented Reality on MECs

To instantiate the proposed architecture fpr the Mobile Augmented Reality application presented in Section 3, the client MAR application must continuously capture frames from the camera and send them together with other parameters (type of POIs of interest, screen size and resolution) to the nearest MEC server. The server is in charge of retrieving the features of the POIs in the scene, match them against a local database, and return the corresponding data (information about monuments, buildings and other points of interest) to the client application, which must merge them with the image on the screen to offer a seamless experience to the user.

Serverless functions deployed on the MEC servers are in charge of: 1) image processing; 2) feature extraction; 3) matching; and 4) information retrieval based on these features. Many of these activities are supported by libraries already integrated in major vendors' serverless frameworks, such as IBM Visual Recognition[4], Azure Visual Cognitive Services[5] and AWS Rekognition[6]. The

[4] https://console.ng.bluemix.net/catalog/services/watson_vision_combined
[5] https://azure.microsoft.com/en-us/services/cognitive-services
[6] https://aws.amazon.com/rekognition/

management of the execution of these functions is optimized by the serverless environment on the MEC server, and different client applications may use the same functions (for instance, those related to image processing and other common use cases).

The MEC architecture further provides the advantage of data locality, which restricts the scope of the feature matching by letting a given MEC server to store data only regarding the POIs within the region covered by its base station (instead of considering the probably wider area covered by the cloud service). Such advantage has two aspects: feature matching against a reduced database becomes less expensive, and substantially reduces data fetching latency [19], and less data must be persisted on each MEC server.

Finally, the creation and update of existing information about the POIs managed by different base stations could be performed by administrators by means of a Web application backed by cloud services. Following this approach, administrators could also request reports about the usage of the MAR application on each base station (e.g., which touristic assets have been most accessed and which advertised services have been most viewed in a given period of time).

5 Experimental Evaluation

We evaluated the proposed architecture in the context of the MAR application (Section 3), using two alternative deployments for the serverless functions: at the edge or in the cloud. The main goal of this experiment is not to compare "traditional" cloud services against a serverless solution, but to demonstrate that the proposed serverless edge architecture can outperform a typical serverless cloud provider under certain circumstances and requirements.

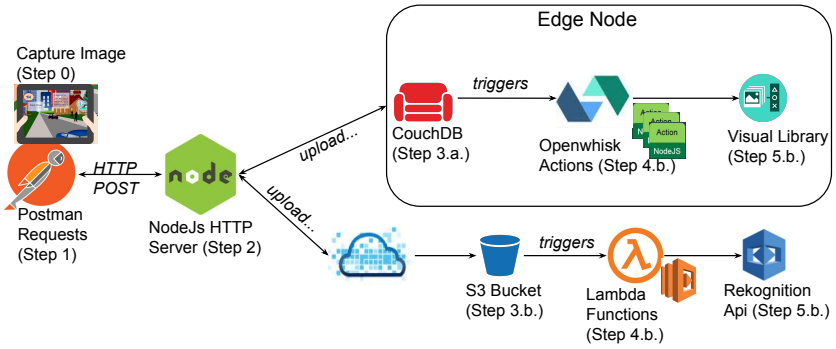


Fig. 3. Experimental Setup for the Example System.

The experimental setup is depicted in Figure 3. Capturing and uploading an image (Steps 0 and 1) is emulated using Postman[7], a JavaScript open source ap-

[7] https://www.getpostman.com/

plication designed to load test functional behaviors and measure the performance of Web APIs.

A Node.js Http server provides the endpoint for the requests and uploads of the image (Step 2), then triggering different subsequent steps depending on the two different deployments: Steps 3.a, 4.a and 5.a for the edge-based solution, and Steps 3.b, 4.b and 5.b for the cloud one. Additionally, the Node.js server collects the metrics relevant to the experiment, such as latency, throughput and computation time.

The edge node deploys the IBM Openwhisk serverless framework[8] that manages *actions* (the equivalent of functions in openwhisk). Being open-source, openwhisk is (to date) the only serverless alternative among the major vendors that can be deployed locally or on private clouds. Particularly, openwhisk provides a built-in noSQL database: CouchDB, which is associated with the implemented actions through user-defined triggers and rules. In our experiment, uploading an image to CouchDB (Step 3.a) triggers the action that performs the feature extraction and matching (Step 4.a) with the points-of-interest, supported by a visual recognition library (Step 5.a).

For this experiment, we considered two alternatives for the deployment of the serverless architecture to mimic the behavior of an edge node (Figure 4). The edge-local alternative is an implementation with openwhisk deployed on a regular laptop, in a virtual machine with 4x CPU, 4x GB of RAM and 40 GB SSD of storage. This deployment allows us to represent an extreme situation where latency is close to zero, but the computational resources are highly constrained. On the other hand, we deployed the serverless architecture on Policloud[9], the private IaaS solution of Politecnico di Milano where the computational resources are less constrained, and still low latency can be achieved due to physical proximity and data locality. This setup runs on a small cluster of 4 virtual machines with 2x CPU, 4x Gb of Ram and 100 GB SSD, each running a different component of openwhisk (triggers and storage, Http server, controller, and invokers). Note that in both cases the edge node is deployed in the same LAN that originates the requests, to emulate the few-hop scenario in which devices are directly connected to their corresponding MEC.

The cloud alternative for this experiment uses AWS Lambda[10] and the associated AWS services, as the first-available and most mature serverless solution in the market. Both the functions and the services (S3 storage, image recognition) are hosted in the us-west region, which is enforced by AWS to guarantee a certain degree of data locality. The image is uploaded through an S3 bucket (Step 3.b), a trigger associates it with the corresponding lambda functions (Step 4.b) that perform the feature extraction and matching supported by the AWS Rekognition service (Step 5.b).

The size of the payload for this experiment was fixed using a sample image of approximately 500 Kb, which is a reasonable size for this use case [20]. The

[8] https://developer.ibm.com/openwhisk/
[9] http://policloud.polimi.it/
[10] https://aws.amazon.com/lambda/

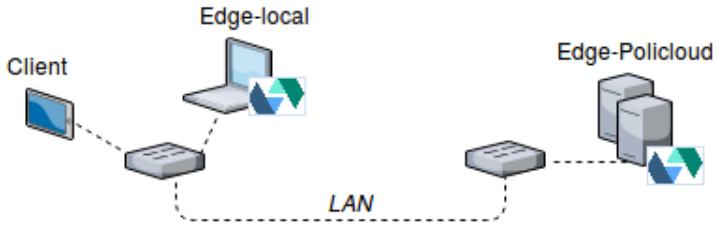
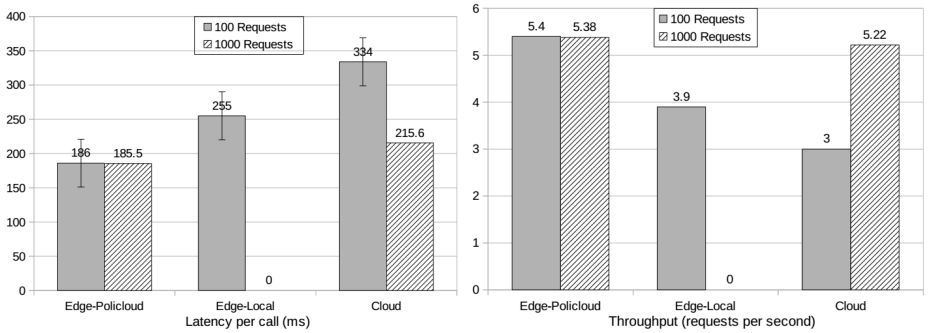


Fig. 4. Deployment alternatives to mimic the behavior and network proximity of edge nodes.

workload was parameterized, ranging from 100 to 1000 requests, considering not only the default maximum for concurrent executions in AWS Lambda[11], but also the limited resources of the local edge node. All functions deployed at the edge and on cloud were configured with a maximum of 256 Mb of RAM per instance.

Results Figure 5 shows the execution results for 100 and 1000 requests served by the edge-based (locally and on PoliCloud) and cloud-based deployment alternatives. We run five times each experiment and show the average values.

Fig. 5. Experimental results for 100 and 1000 requests in the Edge-based and in the Cloud-based deployments.



(a) Latency per call (less is better) (b) Throughput (more is better)

The latency is shown in Fig. 5(a), along with the standard deviation, calculated as the average over 100 and 1000 requests, respectively. These results do not consider the actual computation time of the functions, that is, they only consider the overhead of network communication per call. For the 100-request scenario, the latency added by the edge-based solution is 80% and 31% less (Policloud and local, respectively) than the latency in the cloud alternative. For 1000

[11] http://docs.aws.amazon.com/lambda/latest/dg/concurrent-executions.html

simultaneous requests, Edge-Policloud maintains a latency similar to the previous scenario. It still shows a clear advantage over the Cloud deployment (72% in latency), which features a slight improvement but still higher latency and higher deviation (as shown by the error indicators of top of each bar in Fig. 5(a)). The results for edge-local deployment are not shown since it was not able to serve this heavy workload, thus the openwhisk architecture throttles the execution causing considerable overhead. The throughput is shown in Fig. 5(b) (standard deviation is negligible thus not shown here) where the number of requests served per second is better in the edge-based solutions, 80% (Policloud) and 30% (local) for the 100 requests scenario. Regarding the 1000-request scenario, Edge-Policloud maintains a similar throughput, a 3% better than the Cloud deployment, which improved significantly due to the higher degree of parallelism achieved. Again, the throughput for the Edge-local deployment is not shown since it was not able to serve the workload timely.

Discussion Obtained results confirm our hypotheses regarding the higher latencies introduced by a cloud solution in the context of data-intensive, low-latency applications. Despite the high degree of parallelism that can be achieved by deploying a serverless solution in the cloud, the throughput decreases when dealing with a heavy workload with images as payload.

In the 100-request scenario, where the cloud solution does not exploit all the parallelism that it can achieve, the Edge solution clearly outstands. Particularly, the Edge-Policloud solution outperformed the Cloud one by a 80% both in latency and throughput. Even the Edge-local solution brings some improvement (30%) despite its strictly constrained resources.

In the heavy workload scenario, the throughput of the Edge-Policloud and Cloud solutions are similar. The naïve edge-local alternative fails on this scenario since it cannot increase its allocated resources, which is a potential shortcoming of too resource-constrained MEC nodes. We foresee that with even heavier workloads, the Cloud solution will certainly outperform, since the higher latencies introduced by sending/retrieving data from/to the Cloud are compensated by its high scalability and parallelism, serving almost all requests simultaneously. However, in a real deployment, we foresee that the edge nodes will also have access to more resources than in our experiment. Although the edge is certainly more resource-constrained than the cloud, several edge nodes would be involved and interconnected in this architecture, allowing one to load-balance the requests among them, and this achieve better throughput and lower latency, as shown in the experiments.

Threats to Validity First, the CPU power of serverless functions is allocated proportionally to their memory configuration[12]. Thus, for CPU-intensive applications, allocating the maximum memory to cloud functions will certainly outperform the edge alternative (where it is not feasible to over-allocate memory and CPU due to limited resources) because of the shorter processing times,

[12] https://aws.amazon.com/lambda/faqs/#functions

and mitigates the gains in terms of latency. One should test and benchmark the architecture to find the adequate trade-off among the resources allocated to functions, the resources available in the edge nodes, and the overall cost. Second, the connection among nodes in the mimicked edge architecture (local and Policloud) was done through LAN (as depicted in Figure 4), which may deliver different connection speeds than a cellular network. To make this scenario more accurate, we emulated 4G connection speeds between the Postman requests and the Node.js server (Figure 3) using network throttling tools[13]. Experiments with real mobile devices and different link quality are very important. Finally, the experiments focused on the latency of the serverless architecture stressed with varying numbers of requests to the same functions. The performance of a serverless solution stressed with heterogeneous functions calls was not part of this work. Nonetheless, the ability of serverless providers [11, 6] to handle the deployment of heterogeneous functions on a limited set of containers is a strong argument in favor of our solution when compared against a "simple" container-based edge solution.

6 Related Work

The work in [3] presents the technical details of the first real-world MEC platform by Nokia Siemens and Intel [21]. In this platform, MEC servers on base stations are equipped with commodity hardware and application deployment is based on virtualization technologies. Applications running on the mobile edge are expected to be event-driven, which is in line with the serverless model discussed in our paper. Besides, the authors present a taxonomy of MEC applications that can profit from MEC deployment. Interestingly, our MAR application (Section 3) is representative of two of the most benefited application classes: "Offloading" and "Augmentation".

Ismail et al. [22] evaluated different aspects of the deployment and operation of a container technology locally on edge nodes. In their work, a testbed was setup using a database and three edge nodes interconnected by a company network. Despite the similarity with this work, our proposal moves away from virtualization and containerization of application logic, in favor of serverless computing to optimize the use of edge resources and boost the potential of mobile edge computing.

The work in [4] proposes two different recovery schemes for overloaded or broken MEC servers. One recovery scheme is where an overloaded MEC server offloads its work to available neighbors within transfer range. The other recovery scheme is for situations when there is no available neighboring MEC within transfer range, and uses devices as ad-hoc relay nodes in order to bridge two MEC servers. In a similar direction, Tärneberg et al. [2] proposed a model that bridges mobile edge computing and the distributed cloud paradigm, as well as an algorithm to solve the resource management challenges that emerge from this

[13] https://developers.google.com/web/tools/chrome-devtools/network-performance/network-conditions

integration. In contrast with these works, our approach mitigates the overload in MEC servers by deploying a serverless architecture on them, which provides an effective and efficient usage of available resources. Certainly, the scalability of our proposed architecture could be extended by means of a neighbor offloading strategy as proposed in [4] or by an integration of MEC and cloud resources as proposed in [2].

The first documented efforts for bringing serverless capabilities to the edge are very recent, and come mostly from industry. Lambda@Edge[14] is a new functionality of AWS (in preview at the time of writing this paper) that allows one to explicitly deploy lambda functions to certain edge locations, closer to the user. However, the notion of edge locations in AWS is coarse grained (but finer grained than AWS regions): their edge schema, named CloudFront, consists of approximately 70 edge nodes worldwide. In contrast, we consider that MEC enables fine-grained edge nodes to be deployed closer to the user. In our proposed architecture, MEC servers can be distributed one every km^2 or less. Furthermore, the upcoming small 5G cells and microcells [3] allow us to think of one edge node per block, or even per building in certain vital places, such as government buildings, shopping centers or transport stations.

EdgeScale [23] is another platform that leverages serverless cloud computing to enable storage and processing on a hierarchy of data centers, positioned over the geographic span of a network between the user and traditional wide-area cloud providers. EdgeScale applications are structured as lightweight, stateless functions that can be rapidly instantiated on demand. This approach implements all the functions, storage, routing and additional capabilities from scratch, while we opted for leveraging current open technologies such as Openwhisk, which have broad support from a major vendor (IBM) and an active community. Besides, regarding the expected benefits of the approach, EdgeScale is on an early stage and does not report any empirical evaluation of concrete gains in terms of latency, throughput and bandwidth.

7 Conclusions and Future Work

This paper presents a novel *serverless edge computing* architecture that enables the offloading of mobile computation with low latency and high throughput. MEC servers are ideal candidates for offloading the computation to preserve devices' resources and kill latency, while a serverless model provides inherent scalability and optimal resource utilization as the allocation of functions to containers is handled by the serverless platform itself, and the number of running functions always matches the trigger rate. Additionally, the application developer only focuses on the application code and can fully outsource the management of the deployment/execution infrastructure.

The proposed architecture is instantiated using a Mobile Augmented Reality application, as a good example of a low-latency application in which the latency

[14] http://docs.aws.amazon.com/lambda/latest/dg/lambda-edge.html

introduced by transferring heavy payloads from/to the cloud can degrade the user experience. We conducted experiments comparing an edge-based solution with a cloud-based solution in this scenario, with the former outperforming the latter up to 80% in terms of throughput and latency.

Our future work comprises the scenario in which several edge nodes are interconnected and can be involved in serving the requests. This should allow us to achieve better throughput and lower latency, but with the additional complexity of introducing load-balancing and resource-allocation mechanisms [24]. Additionally, the comparison with a traditional (non-serverless) deployment in the cloud should be addressed, to find the right balance among resource consumption, performance, and cost.

References

1. Dehos, C., Gonzlez, J.L., Domenico, A.D., Ktnas, D., Dussopt, L.: Millimeter-wave access and backhauling: the solution to the exponential data traffic increase in 5g mobile communications systems? IEEE Communications Magazine **52**(9) (September 2014) 88–95
2. Tarneberg, W., Mehta, A., Wadbro, E., Tordsson, J., Eker, J., Kihl, M., Elmroth, E.: Dynamic application placement in the mobile cloud network. Future Generation Computer Systems **70** (2017) 163 – 177
3. Beck, M.T., Werner, M., Feld, S., Schimper, S.: Mobile edge computing: A taxonomy. In: Proc. of the Sixth International Conference on Advances in Future Internet. (2014) 48–54
4. Satria, D., Park, D., Jo, M.: Recovery for overloaded mobile edge computing. Future Generation Computer Systems **70** (2017) 138 – 147
5. Pahl, C.: Containerization and the paas cloud. IEEE Cloud Computing **2**(3) (May 2015) 24–31
6. Roberts, M.: Serverless architectures: What is serverless? (2016) Retrieved from: http://martinfowler.com/articles/serverless.html.
7. Fromm, K.: Why the future of software and apps is serverless (2012) Retrieved from: http://readwrite.com/2012/10/15/why-the-future-of-software-and-apps-is-serverless/.
8. Salman, O., Elhajj, I., Kayssi, A., Chehab, A.: Edge computing enabling the internet of things. In: IEEE World Forum on Internet of Things (WF-IoT). (Dec 2015) 603–608
9. Ahmed, A., Ahmed, E.: A survey on mobile edge computing. In: 2016 10th International Conference on Intelligent Systems and Control (ISCO). (Jan 2016) 1–8
10. Hu, Y.C., Patel, M., Sabella, D., Sprecher, N., Young, V.: Mobile edge computing: A key technology towards 5g. ETSI White Paper **11** (2015)
11. Hendrickson, S., Sturdevant, S., Harter, T., Venkataramani, V., Arpaci-Dusseau, A.C., Arpaci-Dusseau, R.H.: Serverless computation with openlambda. In: Proceedings of the 8th USENIX Conference on Hot Topics in Cloud Computing. (2016) 33–39
12. Villamizar, M., Garcés, O., Ochoa, L., Castro, H., Salamanca, L., Verano, M., Casallas, R., Gil, S., Valencia, C., Zambrano, A., Lang, M.: Cost comparison of running web applications in the cloud using monolithic, microservice, and aws

lambda architectures. Service Oriented Computing and Applications **11**(2) (2017) 233–247
13. Barfield, W.: Fundamentals of wearable computers and augmented reality. CRC Press (2015)
14. Huang, B.R., Lin, C.H., Lee, C.H.: Mobile augmented reality based on cloud computing. In: Anti-counterfeiting, Security, and Identification. (Aug 2012) 1–5
15. Wagner, D., Schmalstieg, D., Bischof, H.: Multiple target detection and tracking with guaranteed framerates on mobile phones. In: IEEE International Symposium on Mixed and Augmented Reality (ISMAR). (2009) 57–64
16. Dollar, P., Wojek, C., Schiele, B., Perona, P.: Pedestrian detection: An evaluation of the state of the art. IEEE Transactions on Pattern Analysis and Machine Intelligence **34**(4) (April 2012) 743–761
17. Baresi, L., Guinea, S., Mendonca, D.F.: A3droid: A framework for developing distributed crowdsensing. In: 2016 IEEE International Conference on Pervasive Computing and Communication Workshops (PerCom Workshops). (March 2016) 1–6
18. Sill, A.: Standards at the edge of the cloud. IEEE Cloud Computing **4**(2) (March 2017) 63–67
19. Abase, A.H., Khafagy, M.H., Omara, F.A.: Locality sim: Cloud simulator with data locality. International Journal on Cloud Computing: Services and Architecture (IJCCSA) **6** (December 2016) 17–31
20. Rodriguez-Santana, B.G., Viveros, A.M., Carvajal-Gámez, B.E., Trejo-Osorio, D.C.: Mobile computation offloading architecture for mobile augmented reality, case study: Visualization of cetacean skeleton. International Journal of Advanced Computer Science & Applications **1**(7) (2016) 665–671
21. Nokia Siemens Networks, Intel: Increasing mobile operators' value proposition with edge computing (2013) Retrieved from: http://www.intel.co.id/content/dam/www/public/us/en/documents/technology-briefs/edge-computing-tech-brief.pdf.
22. Ismail, B.I., Goortani, E.M., Karim, M.B.A., Tat, W.M., Setapa, S., Luke, J.Y., Hoe, O.H.: Evaluation of docker as edge computing platform. In: 2015 IEEE Conference on Open Systems (ICOS). (Aug 2015) 130–135
23. de Lara, E., Gomes, C.S., Langridge, S., Mortazavi, S.H., Roodi, M.: Hierarchical serverless computing for the mobile edge. In: IEEE/ACM Symposium on Edge Computing (SEC), IEEE (2016) 109–110
24. Baresi, L., Guinea, S., Leva, A., Quattrocchi, G.: A discrete-time feedback controller for containerized cloud applications. In: Proceedings of the 2016 24th ACM SIGSOFT International Symposium on Foundations of Software Engineering. (2016) 217–228