Designing a decentralized container based Fog computing framework for task distribution and management

Anna Reale, Péter Kiss, Melinda Tóth, Zoltán Horváth

***Abstract*—In the field of Fog computing, because of the highly dynamic nature of Mobile Networks, the problem of automating task distribution and code migration has not yet been solved by any resource orchestrator. In this paper, we propose a solution to the problem in the form of an orchestrator that can build overlay computational networks from a volatile set of nodes. Each node of this overlay is able to deploy specific services and can satisfy the requested task from any participant node.**

We also demonstrate the infrastructure purpose by suggesting an existing Federated Learning Application.

***Index Terms*—Fog computing, ad-hoc networks,orchestration, docker containers, Federated Learning.**

I. Introduction

SERVICE and network management for Fog computing is a complex subject. On one side we have the device heterogeneity and, on the other, the different control the network owner has on each network node. In fact, user-owned devices can participate actively in the service as well, by running part of the required applications. For these reasons, a centralized solution cannot work efficiently. Thus, other technologies have been suggested in the literature [1], for instance: software-defined networks; asymptotic techniques for scaling management (e.g. declarative states as opposed to individual commands for admin purposes); edge clouds; and peer-to-peer (P2P) and sensor network-like approaches for auto-coordination of applications.

Main contributions of this work are the definition of a decentralized container based Fog computing framework for task distribution and management (in Section II) and the example implementation (in Section III) based on the use case constraint described in the following paragraphs (I-A and I-B).

A. FOG

With the term Fog computing, we refer to a highly virtualized platform that provides services related to computation, storage and networking and that can be seen as a layer in between the Data Centers and the final user equipment/devices. This concept was introduced in paper [2]. Since the original definition leaves the Fog paradigm very close to other Edge Computing ones as Multi-access Edge Computing [3] and Cloudlets [4], in this work we adopt Vaquero [5] views on Fog computing. The author distinguishes Fog from other edge computing architecture by specifying that the involved nodes may be in a great quantity and extremely heterogeneous (wireless and sometimes autonomous). Vaquero envisions Fog nodes that can deploy decentralized communication and potentially cooperate among each other to perform storage and processing tasks in an autonomous fashion. Nodes tasks can range from basic network functions support to new services and applications running in a sandbox environment. Thus, the most interesting characteristic of this interpretation would be that users can become an active part of the Fog network, by not only being consumers but also providers: leasing part of their devices to host services and get incentives for doing so.

The main factors that will bring the fog can be summarized as follows:

- *Nodes edge location and geographical distribution*: Fog nodes must be deployed at the edge of the network and must be able to locate at least their neighbours in the local area. Due to the geographical distribution and the closeness to the final user, they will be able to communicate at low latency and reduce in data movement across the network significantly.
- *User mobility*: fog applications can communicate directly with mobile devices via mobility techniques and protocols such as LISP [6] (Cisco's Locator/ID Separation Protocol) that can decouple host identity from location identity [7].
- *Nodes heterogeneity*: Fog is a multi-layered hierarchical infrastructure, with dynamic and miscellaneous nodes deployed in a wide variety of environments, possibly in both physical and virtual form.
- *Nodes interoperability*: Fog nodes must be all-purpose and able to inter-operate even if related to different providers' networks.
- *Nodes and users real-time interaction*: services with low latency, and involving continuous real-time interactions between users and system.

B. The use-case

In our solution, we aim at a general framework to enable single network nodes to outsource computationally or resource intensive tasks. Following the idea of Fog computing, in which individual nodes collaborate at the edge of the mobile network in solving tasks that would exceed the capabilities of single

ELTE Eötvös Loránd University, Budapest, Hungary
Faculty of Informatics, 3in Research Group, Martonvsr, Hungary
{anna.reale, peter.kiss, toth_m,hz}@inf.elte.hu
ORCID: 0000-0001-8295-2782, 0000-0001-6941-2095, 0000-0001-6300-7945, 0000-0001-9213-2681
Manuscript received March 29, 2019;

local nodes, or would be overly expensive (in terms of time, resources or money) if performed by nodes physically further in the network.

As a rather extreme, concrete and simple scenario, one can think of mobile phones used to train machine learning models for various purposes, like speech recognition, or auto-completion for texting. The most straightforward approach to train such models, where the training data is generated in a massively distributed fashion, is to transmit the training data to powerful data-centers, where the training process occurs. Above the obvious networking overhead, this technique gives rise to some ethical dilemmas related to users privacy. The above use-case corresponds to the trend of federated learning [8], [9], [10].

In the use-case we can differentiate the initiator, that can be a company providing an application that uses the learned model, and the users who will profit from the ML model trained in collaboration.

In this work primarily we follow the perspective of the initiator, individual users join the network, cooperatively run parts of the training process, and then use the common knowledge learned and published by the initiator.

C. Paper terminology

- **Initiator** denotes a software or person who wants to solve a specific problem.
- **Service** refers to the software to be run. We can look at the ensemble of outsourced/deployed modules as a service the network grants to the initiator. (And on the other hand, it will be analogous with the terminology used at swarms.)
- **Task** is the part constituting a module of the service.
- **Host** covers the physical machines where the task will be eventually executed.
- **Node** is a uniform application run on a host, essentially these compound together with the infrastructure for service deployment.

II. The framework

The proposed system is a peer-to-peer collaborative computation network. The nodes in the network are light-weight uniform piece of software, whose main responsibilities are: to maintain the graph of the available nodes, to move the code of the pieces of the deployed application and to organize communication among the deployed modules (as represented in Fig. 1). The framework can be regarded as a decentralized extension to the existing distributed container technologies.

Applications that are intended to be run over the network, must be partitioned to minimize the imposed overhead. An idea could be to separate monolithic applications along the minimum cuts of their function call graph (or data flow graph or in some other balanced middle-way), to minimize data flow in between parts [11].

In this work we regard applications as a composition of micro-services, assuming that with more or fewer efforts any application can be transformed into such an architecture.

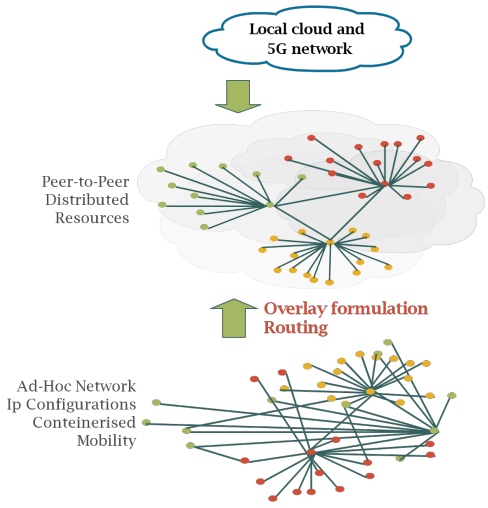

Fig. 1. Our fog network implementation is perceived from the rest of the mobile network as multiple overlays deploying different services

In such a framework we had to address in the design process: (A) the mapping of the available resources, (B) the creation of the deployment plan, (C) the plans and approaches to move codes to their assigned location, and (D) the strategy to maintain the health of the service and the shared state of the application, that is how to transfer data between the functions, and how to keep globals up-to-date.

A. Network mapping

When in need to deploy or execute a service, a network user can make a decision on the deployment according to the mapping strategies best fitting the characteristics of the service. To make it possible, the system should provide the initiator node information about the possible Service Level Agreements (SLAs): the description of the available resources along with the price of the usage of a particular resource, also expressed in terms of latency or volatility of the perceived QoS (Quality of Service). We will further discuss possible parameters to be considered in the next paragraph.

B. Task assignment

With the mapping of the available network and the call-graph/interaction graph of the service, the next task is to map the modules to nodes. There are several methods to do task assignments, like the placement of Virtual Machines (VM) in cloud computing or placement of Virtual Networks Function (VNF). The paper [12] shows a global classification of VM placement solutions into online and offline approaches, categorizing them based on their target objective. Some of the solutions are single objectives, others may have several. The point of view of the analysis may be either the final user or the network providers. Among the objectives we can mention:

- Energy consumption: based on minimizing power consumption and number of active nodes;
- Cost: as Return On Investment (ROI), resource exploitation or allocation cost;

- QoS: can be expressed in terms of response time, overhead time;
- Resource usage: RAM, CPU, storage;
- Reliability;
- Load balancing: avoidance of congestion, data overload.

C. Deployment and Service Migration Infrastructures

Given our premises on Fog, an important task, for improving service deployment, appears to be not only defining at what granularity a given service should be divided among some of the nodes, but also what is the best way to encapsulate the application that it consists of. This will permit service division and migration following the user movement and the rules of geographical distribution and load balancing.

At deployment, thus, the challenge is how to move the code and resources between the nodes. As a base scenario, we assume that the modules of the service to be deployed resides at the initiator. The initiator can contact the joined nodes assigned to the tasks and push the codes and data that he needs to run. Multiple applications, each made of several components should simultaneously use the Fog infrastructure. Memory isolation is necessary for security and integrity reasons but also for bug prevention and performance tuning. The real-time constraint is an added requirement to the migration.

Finally, the nodes of the described strongly ad-hoc system might run over different platforms, therefore running tasks of the service in a virtualized environment seems inevitable.

In the following paragraph, we will compare two possible technologies to achieve the described objectives: virtual machines (VM) and container-based architecture.

1) Containers over VM for migration in a Fog: A virtual machine based solution involves a hypervisor and can be resource intensive and more complex to scale compared to containers. In fact, each VM would require its own full OS, TCP and file system stacks, significantly impacting the processing power and memory of its host [13].

Since containers rely on the hosting operative system, they are more compact in size and lighter to migrate thanks to their layered architecture. In fact, in a container, some layers are read-only while one is a read and write layer. When migrating, only this read and write layer needs to be moved [14]. In other words, once a container is installed, only the extra different layers, such as additional binaries and libraries, needs actual migration. Installations involve the image of the container, comprised of system libraries, system tools and other platform settings a software program needs to run on a containerization platform.

Furthermore, containers do not waste RAM on redundant management processes, generally consuming less RAM than a VM. Each VM has a fixed amount of RAM, and we usually reserve resources in each VM for the further scaling of the application. These resources are not fully utilized, and, at the same time, they cannot be shared with other applications due to the lack of proper instance isolation.

Static memory allocation for virtual machines (VMs) can lead to severe SLA violations or inefficient use of memory [15]. Few hypervisors can resize VMs while running with the help of dynamic memory allocation mechanisms such as ballooning [16] and memory hotplug [17] were proposed to handle the dynamics of memory demands.

Instead, resource limits in containers can be easily changed on the running instances without a restart. And the resources that are not consumed within the limit boundaries are automatically shared with other containers running on the same hardware node.

Taking into account these differences and the fact that Fog nodes are diverse, often with limited bandwidth, unstable network connectivity, costly or limited storage and processing capability, running a container-based solution will be much more beneficial.

As mentioned in [18], with containers, the complexity can be reduced through container abstraction since they avoid reliance on low-level infrastructure services. Automation can be supported to maximize portability. Security and governance can be achieved by placing services outside the containers. Higher computing capability can be provisioned with service composition, achievable even if the containers run on different physical machines.

2) Existing Containers Migration Technologies: Given the above mentioned advantages, more and more mainstream operating systems begin to adopt container technology to provide isolation and resource control, which has demonstrated great potential for service migration [19]. Live migration of containers is now possible using CRIU [20](Checkpoint Restore In Userspace) that supports checkpoint and restore functionality for Linux. Nowadays CRIU supports the checkpointing and restoring of containers for OpenVZ, Linux Containers (LXC) and Docker. The main downside of OpenVZ [21] is the usage of a shared file system among the nodes, this solution is not adaptable in a fog scenario.

LXC is a userspace interface for the Linux kernel containment features. Through a powerful API and simple tools, it lets Linux users easily create and manage system or application containers [22].

Docker [23] extends LXC with a kernel- and application-level API that together runs processes in isolation: CPU, memory, I/O, network, and so on. Docker also uses namespaces to completely isolate the application view of the underlying operating environment, including process trees, network, user IDs, and file systems [24].

Docker containers are created using a base image: it can include just the OS fundamentals, or it can consist of a sophisticated application stack. When building images each action taken forms a new layer on top of the previous one.

Previous works proved slightly better performance and flexibility for Linux Containers compared to Docker [25]. The authors privileged Docker mostly because of ease of use: Docker seems a more likely commercial and user-friendly solution. Also, Dockers are standalone applications running in an isolated environment, thus better respecting our necessities.

D. Maintaining the health of the network

After deploying an asymptotically optimal architecture in the overlay network, the main task is to keep the quality

of service at an acceptable level. In a Fog computing environment, an obvious challenge is the strong variance of the communication channels or the workload at the nodes, that makes relocation of tasks necessary.

The workload may change also due to nodes abandoning or joining the network. This high turnover rate makes redundancy necessary: duplicating processes over statistically more reliable nodes like those more strongly connected to the fixed network of the network provider.

1) Open Source Containers Orchestration: As far as management platforms go, the two most used solutions for Docker engines are Kubernetes or Docker Swarm. Kubernetes is currently considered a standard platform. Kubernetes works around the concept of pods, which are scheduling units (able to deploy one or more containers) in the Kubernetes ecosystem. Pods are distributed among nodes to provide high availability. It presents plenty of helpful resources, and guidance available thanks to its online community.

In Kubernetes each pod deserves its own, unique IP address that is reachable from any other pod in the cluster, whether they are co-located on the same physical machine or not [24].

On the other hand, Docker Swarm has the advantage of being tightly integrated into the Docker ecosystem, using its own API. Swarm has a filtering and scheduling system allowing selection of optimal nodes in a cluster to deploy containers.

A major downside for Kubernetes is not being a complete solution and requiring custom plug-ins to set up. Instead, in Swarm, dependencies are handled well within the ecosystem, making installation and setup user-friendly. However, most cloud providers today offer Kubernetes as a service. Finally, fault tolerance handled in a better way in Kubernetes.

2) Migration Support: To our knowledge, Kubernetes does not presently support live migration of its pods and the feature is not actively being worked on. Pods are scheduled, started, and eventually terminate. They are replaced with new pods by the replication controller. Today pods are replaced reactively, but eventually, it will replace pods proactively for planned moves [26]. Currently, new pods have no obvious relationship to the original pods they replace. They have different names, different user ids, different IP addresses, different hostnames (since we set the pod hostname to pod name), and newly initialized volumes.

Migration strategy in Kubernetes seems to be cold: it shut down, move and restart, losing any intermediate state.

As mentioned before, Swarm could potentially use docker experimental feature integrating CRIU basic live migration. However, this has to be completely configured and handled by the user.

Currently, Docker can deploy a hot migration: using the experimental mode, one can suspend the container, move it to the destination host and resume the computation while preserving the persistent applications state.

III. Experimental Settings

In this paper we make an attempt to describe a potential model implementation of a Fog computing environment fulfilling the requirements described at Section II, using existing and more or less mature technologies as *Docker*, *Docker Swarm*, *CRIU* and *runc*. The proposed system adds an additional orchestrator over the combination of the mentioned systems. The orchestrator enables dynamic adjustments on the underlying infrastructure where the managed service is running on.

A. Network mapping

Before the deployment phase, a peer sends a request to the neighbouring nodes with a unique identifier. As a node receives this request it returns an acknowledgement, through which the actual latency can be learned. A receiver then forwards the request to its neighbours, who are statistically predicted to be within the remaining latency constraint, measures the actual latency and so on. If a peer has already received a mapping request with the same id, it only sends back the acknowledgements since the sub-network that is reachable through it is already submitted in another mapping path. This helps to avoid network overflowing with useless traffic. The results will be sent back recursively to the requiring node, eventually reaching the originator, where a local picture of the neighbouring nodes will be assembled.

The mapping could be carried out in two phases. The first randomized phase is about maintaining the shape of the network along with some important statistics of the available resources of the nodes and the quality of the links between them. Based on this later on, when the system makes the deployment plan it can incorporate the learn knowledge on the volatility of the availability of the components.

B. Task assignment

Task assignment phase covers the creation of a deployment strategy for the tasks in the service, that is a mapping between the physical network and our task call graph, which is the result of partitioning the software to be deployed. The topic of this task assignment is often referred to as Path Computation and Function Placement and is described in [27]. For this scope, we selected an eager randomized approach of randomized rounding introduced in [27].

According to this method, the service chain graph, representing the application tasks and their interaction, is traversed in Breadth-First Search fashion, assigning the root task to the initiator node. At edges in the graph (calls between the modules/tasks), the location of the subsequent task will be chosen pseudo-randomly from the neighbouring nodes of the source task node. This happens so that hosts with higher capabilities and/or better communication links from the node of the parent task will be chosen with a higher chance. Different policies can be provided. A more detailed description of the process is given in [11].

C. Deployment

The mentioned need for interoperability urges the use of virtualization techniques. Based on the motivation stated in Section II-C2, we choose Docker containers to encapsulate tasks.

To reduce the complexity of routing the communication between tasks located at different hosts, the containers carrying

the code of the tasks will be deployed to a Docker swarm, assigning a unique port to each task. At the deployment, the initiator node of the service creates a swarm and sends a request to the nodes that are the assigned location of the individual tasks to join the swarm. The way the source containers are created at the nodes depends on the caches of the node.

At the first deployment of the service, when no images exist for the tasks, the efficient way to move and execute codes at deployment is to move the all the service-specific volumes and build the images and containers at the destination node. In this case, the source code of a task along with the data needed at the execution is passed by the nodes through TCP sockets.

D. Maintaining the health of the network

In a fog environment, one can always expect that the efficiency of the built service will drop. There are various reasons for that, for example, the overloading of a given host or network channel. In the following, we describe the defined orchestrator infrastructure and the selected migration support strategies.

1) Management Orchestration layer: The Management and Orchestration layer is responsible for building up and maintaining an overlay network, that is capable of running the service at predefined QoS (as shown in Fig. 1). The orchestrator consists of a set of peer applications that run on each physical machine. When a node joins a network, it contacts an old node whose address could be obtained through trackers. The old node then shares some of its own connections, to make the network more connected. When a node needs to submit a task it has to map the available network. The notion ”availability” includes the feasibility of the QoS requirements of the given task, in the meaning of taking into consideration the bandwidth requirements and acceptable total network latency.

Technically these nodes should be implemented in a cross-platform language. Thus, we have chosen Java for this purpose. The control communication, that is sending connection or mapping requests, or making agreements on secondary channels uses REST protocols over a standard TCP connection. The nodes maintain the addresses of the nearby nodes along with network segment reachable within a given time constraint.

2) Migration Support: The problem of relieving the workload of struggling hosts, or reducing the increased communication latency can be addressed through tasks relocation. The best strategy we have found to approach the idea of *live migration* is the incremental checkpointing of the state of the task to be removed. To implement iterative migration, we rely on *runc* containers, that are natively integrated with *CRIU*.

According to this, live migration of the container takes the following steps (Figures 2, 3 and 4): (1) through the management nodes we transmit the docker image and the task-specific static volumes [28] (files, that are mounted into the container), that is, the docker image containing the kernel and libraries the task needs. (2) If there are files in the volumes, that are written by the task, they might be synchronized using tools like rsync [29], that provides a very fast method for remote file synchronization. (3) When the above steps

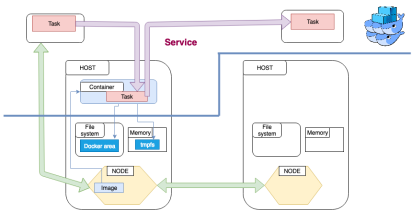

Fig. 2. Before migration

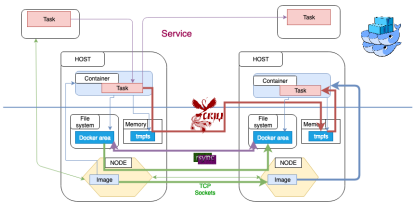

Fig. 3. Migration

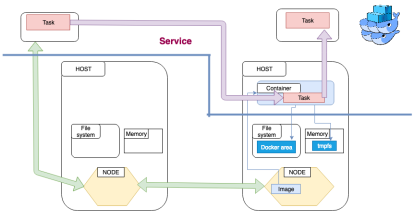

Fig. 4. After migration

are done, it is possible to create the container at the target destination.

What left is to transmit the internal state of the migrated process. For achieving the least process downtime possible, one can execute incremental checkpointing [30] of the process to be relocated. At first, (4) one or more pre-dump needs to be taken, which keeps the original process alive, while starts tracking memory changes. After a number of these pre-dump operations, when everything is ready on the receiver side, the last dump can be taken (5), that kills the process at the original location.

The shipping of the dumps of the old container should take place using the CRIU page server [31], that is a component that has been developed to move user memory to a destination system. This enables disk-less transmission of the state, loading the taken dumps into the so-called *tmpfs* [32] mount at the destination container. The *tmpfs* mounts make possible to write temporary files in the system memory. For that (6) the system starts the container at the destination host and mounts the volume and the *tmpfs*, where the dumps from the old node will be saved.

After the transmission of the final dump, the state of the old task can be restored at the new one. When the migrated process is up, the neighbouring tasks in the call graph will communicate with the new tasks. Since the described system uses Docker swarm, there is no need to rewire the connections,

because it grants us relocation transparency.

Meanwhile, on the original node, it is possible to detach the node from the swarm, since the dump operation stopped the process of it.

IV. Related Works

In the last years, the interest over service placement techniques has piqued, because of the many network paradigm that has arisen especially in connection with 5G and Edge Computing.

Among the works on Service Migration for Edge Computing the paper [18] discusses two concepts similar to service migration: live migration for data centers and handover in cellular networks. The authors present an extensive taxonomy on the topic. They favor Agent-based solutions over VM and containers but stress how these technologies are still at a too early stage.

Also [12] shows a global classification of VM and VFN placement solutions categorizing them based on their target objective. The point of view of the analysis may be either the final user or the network providers. Further analysis of the topic can be found also in our previous work on service placement [11].

Recent works have been done also in the direction of applying containers for service migration. In [33], after a systematic study of Docker container layer management and image stacking, the authors propose a migration method that reduces file system synchronization overhead, without dependence on the distributed file system. The first evaluation result in a reduction of the total service handoff time by 80% with network bandwidth 5Mbps.

In [25] three different mechanisms are proposed and evaluated to improve the end user experience by using container-based live migration paradigm. The author privilege LXC over Docker because of their lighter structure. The described methods are the classical approaches for VM and distributed systems: temporary file system (tmpfs) and disk-less based lightweight container migration, and the shared file system.

Basic notions on the topic of containers checkpointing and live migration can be found in [19]. Also, [34] summarizes the differences between LXC and Docker and introduce the success of Kubernetes.

Related to our use case, in [9] a comprehensive description has been given on the advantages of the federated learning approach, with a range of applicable algorithms. In [8] the authors present a practical method, named Federated average, and through a series of experiments, they show that the idea of federated learning can be effectively and efficiently used in similar scenarios to the ones described in Section I-B.

V. Conclusion and Future Works

Current orchestrators for mobile networks cannot handle the complexity added by new paradigms such as Fog computing. Having to integrate and manage such peer-to-peer and ad-hoc solutions requires the introduction of: (A) mapping of available and mutable resources, (B) creation of a dynamic deployment plan, (C) infrastructures to guarantee codes and application location assignment, and (D) the strategy to maintain the health of the service; especially the shared state of the application.

In this paper we described some of the current technology, their benefit and how we combine them, to build a framework tackling all four of these needs. To illustrate a possible application, we described a Federated Learning case scenario. At present the orchestrator functionalities Network mapping,Task assignment and Deployment are fully implemented. In the next iteration, we would like to optimize the current migration strategy, based on experimental Docker CRIU calls, with what described in Migration Support (Section III-D2).

Acknowledgment

The research has been supported by the European Union, co-financed by the European Social Fund (EFOP-3.6.2-16-2017-00013, Thematic Fundamental Research Collaborations Grounding Innovation in Informatics and Infocommunications).

References

[1] K. Kai, W. Cong, and L. Tao, "Fog computing for vehicular ad-hoc networks: paradigms, scenarios, and issues," *the journal of China Universities of Posts and Telecommunications*, vol. 23, no. 2, pp. 56–96, 2016.

[2] F. Bonomi, R. Milito, J. Zhu, and S. Addepalli, "Fog computing and its role in the internet of things," in *Proceedings of the first edition of the MCC workshop on Mobile cloud computing*. ACM, 2012, pp. 13–16.

[3] E. T. S. I. 2, *Multiaccess Edge Computing (MEC); Technical Requirements*, ETSI ETSI ETSI GS MEC-IEG 004 V1.1.1 (2015-11), 2015.

[4] M. Satyanarayanan, P. Bahl, R. Caceres, and N. Davies, "The case for vm-based cloudlets in mobile computing," *IEEE pervasive Computing*, no. 4, pp. 14–23, 2009.

[5] L. M. Vaquero and L. Rodero-Merino, "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, 2014.

[6] D. Farinacci, V. Fuller, D. Meyer, and D. Lewis, "The locator/id separation protocol (lisp)," CISCO, Tech. Rep., 2013.

[7] P. Raad, S. Secci, D. C. Phung, A. Cianfrani, P. Gallard, and G. Pujolle, "Achieving sub-second downtimes in large-scale virtual machine migrations with lisp," *IEEE Transactions on Network and Service Management*, vol. 11, no. 2, pp. 133–143, 2014.

[8] H. B. McMahan, E. Moore, D. Ramage, S. Hampson, and B. A. y Arcas, "Communication-efficient learning of deep networks from decentralized data," in *AISTATS*, 2017.

[9] J. Konecný, H. B. McMahan, D. Ramage, and P. Richtárik, "Federated optimization: Distributed machine learning for on-device intelligence," *CoRR*, vol. abs/1610.02527, 2016.

[10] Google AI Blog., "Federated Learning: Collaborative Machine Learning without Centralized Training Data." [Online]. Available: "https://ai.googleblog.com/2017/04/federated-learning-collaborative.html"

[11] A. Reale, P. Kiss, C. Ferrari, B. Kovács, L. Szilágyi, and M. Tóth, "Application functions placement optimizaton in a mobile distributed cloud environment." *Studia Universitatis Babes-Bolyai, Informatica*, vol. 63, no. 2, 2018.

[12] A. Laghrissi and T. Taleb, "A survey on the placement of virtual resources and virtual network functions," *IEEE Communications Surveys & Tutorials*, 2018.

[13] A. M. Joy, "Performance comparison between linux containers and virtual machines," in *2015 International Conference on Advances in Computer Engineering and Applications*. IEEE, 2015, pp. 342–346.

[14] Z. Tang, X. Zhou, F. Zhang, W. Jia, and W. Zhao, "Migration modeling and learning algorithms for containers in fog computing," *IEEE Transactions on Services Computing*, pp. 1–1, 2018.

[15] H. Liu, H. Jin, X. Liao, W. Deng, B. He, and C.-z. Xu, "Hotplug or ballooning: A comparative study on dynamic memory management techniques for virtual machines," *IEEE Transactions on parallel and distributed systems*, vol. 26, no. 5, pp. 1350–1363, 2015.

[16] P. Barham, B. Dragovic, K. Fraser, S. Hand, T. Harris, A. Ho, R. Neugebauer, I. Pratt, and A. Warfield, "Xen and the art of virtualization," in *ACM SIGOPS operating systems review*, vol. 37, no. 5. ACM, 2003, pp. 164–177.
[17] S. S. Pinter, Y. Aridor, S. Shultz, and S. Guenender, "Improving machine virtualization with'hotplug memory'," in *17th International Symposium on Computer Architecture and High Performance Computing (SBAC-PAD'05)*. IEEE, 2005, pp. 168–175.
[18] S. Wang, J. Xu, N. Zhang, and Y. Liu, "A survey on service migration in mobile edge computing," *IEEE Access*, vol. 6, pp. 23 511–23 528, 2018.
[19] A. Mirkin, A. Kuznetsov, and K. Kolyshkin, "Containers checkpointing and live migration," in *Proceedings of the Linux Symposium*, vol. 2, 2008, pp. 85–90.
[20] C. Feb, "Checkpoint/restore in userspace," https://criu.org/Main_Page, 2019.
[21] O. Feb, "Openvz," https://wiki.openvz.org/Main_Page, 2019.
[22] L. Feb, "Linux container," https://linuxcontainers.org, 2019.
[23] Docker, "Docker: Enterprise Container Platform." [Online]. Available: "https://docker.com/"
[24] D. Bernstein, "Containers and cloud: From lxc to docker to kubernetes," *IEEE Cloud Computing*, vol. 1, no. 3, pp. 81–84, Sep. 2014.
[25] R. A. Addad, D. L. C. Dutra, M. Bagaa, T. Taleb, and H. Flinck, "Towards a fast service migration in 5g," in *Proceedings of the IEEE NetSoft Conference, Montreal, QC, Canada*, 2018, pp. 25–29.
[26] D. K. Rensin, "Kubernetes-scheduling the future at cloud scale," RedHat,Google, Tech. Rep., 2015.
[27] G. Even, M. Rost, and S. Schmid, "An approximation algorithm for path computation and function placement in sdns," in *International Colloquium on Structural Information and Communication Complexity*. Springer, 2016, pp. 374–390.
[28] Docker, "Use volumes." [Online]. Available: "https://docs.docker.com/storage/volumes/"
[29] rsync, "rsync features." [Online]. Available: "https://rsync.samba.org/features.html"
[30] CRIU, "Incremental dumps." [Online]. Available: "https://criu.org/Incremental_dumps"
[31] ——, "Disk-less migration." [Online]. Available: "https://criu.org/Disk-less_migration"
[32] Docker, "Use tmpfs mounts." [Online]. Available: "https://docs.docker.com/storage/tmpfs/"
[33] L. Ma, S. Yi, and Q. Li, "Efficient service handoff across edge servers via docker container migration," in *Proceedings of the Second ACM/IEEE Symposium on Edge Computing*. ACM, 2017, p. 11.
[34] D. Bernstein, "Containers and cloud: From lxc to docker to kubernetes," *IEEE Cloud Computing*, vol. 1, no. 3, pp. 81–84, 2014.

Anna Reale is a P.h.D. student at the Eötvös Loránd University, with a background in Information Engineering and Computer Science. After her EIT Digital double master degree in Service Design Engineering, she joined an EIT Digital Industrial PhD program In ELTE. Her PhD topic being 5G Edge Computing, she works on representation of nodes resources, context awareness for migration, computation offloading and partitioning frameworks.

Péter Kiss is a P.h.D. student at the Eötvös Loránd University, with a background in Computer Science. After his EIT Digital double master degree in Service Design Engineering, he joined an EIT Digital Industrial PhD program In ELTE. His PhD topic is Distributed Machine Learning and he works on distributed collaborative machine learning, optimization and supporting infrastructures.

Melinda Tóth works as a senior lecturer at the Eötvös Loránd University, teaching distributed systems and Erlang OTP technology. Melinda Tóth is chief architect of RefactorErl, a static source code analysis and transformation system for Erlang. She also works as a researcher at ELTE-Soft Nonprofit Ltd., leading the ELTE-Ericsson Software Technology Lab.

Zoltán Horváth is a full-time professor at the Faculty of Informatics, Eötvös Loránd University, leading the Department of Programming Languages and Compiler. He is also the dean of Faculty. He is teaching and researching functional programming and formal methods for distributed systems. Zoltn Horvth supervised numerous national and international projects, among others he was Principal Investigator in the Parallel Patterns for Adaptive Heterogeneous Multicore Systems (ParaPhrase) EU FP7 project. He is also a project manager at ELTE-Soft Nonprofit Ltd. (Budapest, Hungary).