# On Using Micro-Clouds to Deliver the Fog

Yehia Elkhatib, Barry Porter, Heverson B. Ribeiro,
Mohamed Faten Zhani, Junaid Qadir, and Etienne Rivière

***Abstract*-The cloud is scalable and cost-efficient, but it is not ideal for hosting all applications. Fog computing proposes an alternative of offloading some computation to the edge. Which applications to offload, where to, and when is not entirely clear yet due to our lack of understanding of potential edge infrastructures. Through a number of experiments, we showcase the feasibility and readiness of micro-clouds formed by collections of Raspberry Pis to host a range of fog applications, particularly for network-constrained environments.**

***Keywords*-fog computing, micro-clouds, cloud computing, Internet of Things, distributed systems**

## I. Introduction

Fog computing is coming. The paradigm allows devices at the edge of the network to become proactive in hosting as well as consuming data and services [1]. This is great for interconnecting the swarm of edge devices: wearables, sensors, smart traffic controllers, interactive displays, etc. Perhaps more importantly, the fog offers potential to provide Internet services in locations that have poor access to network and computational resources, as is the case in many developing regions.

Current literature focuses on the fog's vision and high-level potential [1][6] but not the pragmatic means of deploying fog solutions. We identify micro-clouds as platforms that offer promising opportunities in edge resource provisioning. We demonstrate through a series of experiments how such platforms are capable of supporting fog solutions and we give an overview of the state of the art, charting some short- to medium-term challenges.

## II. What are Micro-clouds?

### A. Predecessors

Cyber-foraging and cloudlets have been proposed primarily for mobile offloading [2]. Such proposals were clearly designed for dedicated (hence static) and powerful servers, potentially provided by ISPs. They employed virtual machines (VMs) that are rather heavyweight for limited-capability, potentially transient, edge resources. VMs also tend to grow into immutable and brittle units that are difficult to manage and costly to migrate.

The *droplets* architecture bridges between the centralisation of the cloud and the opposite extreme decentralisation, termed *the mist* [3]. Despite explaining high level mechanics and associated trade-offs, the paper did not specify how such droplets would be deployed. Our experimental results strongly suggest micro-clouds as ideal deployment vehicles for such a vision.

### B. Rise of Micro-clouds

The recent development of small, cheap, low-power computers has prompted a number of new applications, e.g. programmable home automation and entertainment systems. Several projects have taken advantage of this and started assembling a number of such devices to create small computing clusters. The availability of this hardware,

coupled with advancements in technologies that enable multi-tennant resource slicing, facilitated the advent of *micro-cloud systems*.

Micro-clouds are standalone computational infrastructures of small size that can be easily deployed in different locations. Their scale is in stark contrast to that of cloud data centers, yet they can offer similar capabilities in the qualitative terms of access to resources in an on-demand, pay-as-you-go fashion. As such, they represent prime candidates for hosting fog services.

Micro-clouds are also distinct from *mini-clouds* or *mini-data centers*, which are modular server racks deployed indoors, e.g. in a temperature and humidity controlled server room. Micro-clouds also refer to a modular assembling of computers but with the key difference of being easily portable *and* independent of existing infrastructure. Consequently, micro-clouds lend themselves to deployment outdoors as well as indoors, and especially in unprepared or hostile environments.

## III. WHY MICRO-CLOUDS?

We discuss two contrasting use cases where micro-clouds are becoming increasingly important.

### *A. Resource Poor Environments*

Over the years, developers have collectively come up with distributed systems that are essentially modern variants of the classical client-server model. Cloud applications predominantly operate in this fashion: a client device with limited processing and storage responsibilities, communicating with a powerful back end system hosted in a remote data center.

Even in regions where average income is relatively low, end user devices have become fairly affordable for a large fraction of the world population. Despite such hardware becoming increasingly powerful and resourceful, these clients still heavily rely on the back end (the 'server').

Focusing on the location of the 'server', we plot in FIGURE **1** the locations of data centers of major cloud service providers as of February 2016. We also identify the locations of population centers with more than 750,000 people [4] with black dots as indicators of significant market potential.

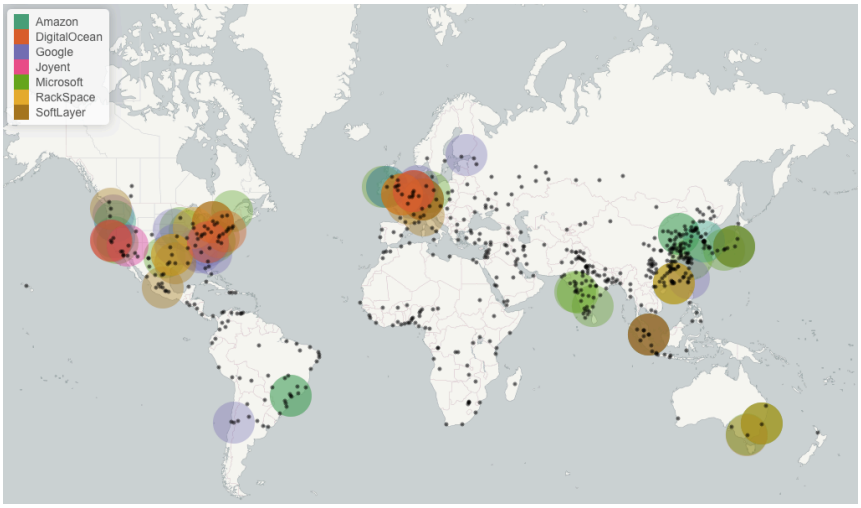



FIGURE 1 - Major cloud data centers (with a 750 mile radius) and major urban areas (black dots), showing that for a large amount of end users no nearby data center is found.

As is evident from FIGURE **1**, for many the 'server' is much further away than desired. This is amplified by the increasing reliance on interaction between users for business and social purposes. The main concern in this model thus becomes the quality of the network connection between clients and the data centers serving them [5].

A solution to this problem is to introduce computational resources where needed. Data centers are indeed being built, but this is a long term solution involving large budgets, high levels of expertise, and national or regional political guarantees. In contrast, injecting smaller infrastructures in the form of micro-clouds provides a much lower cost alternative. They require far less expertise, energy, housing, and geopolitical commitments.

### *B. Resource Rich Environments*

The low cost and easy-to-set-up aspect of micro-clouds render them highly amenable to a number of applications in resource-rich environments. One example is IoT deployments (smart cities, home automation, data-driven industries, etc.) where there is a plethora of sensors and actuators. Micro-clouds promise a number of opportunities in terms of playing support roles such as aggregation, pre-processing, caching, fault mitigation, and migration assistance. This manifestation is supportive of the fog [1][6] and edge-centric [7][8] concepts where heterogeneous devices dispersed around the edge intercommunicate to both provide and consume services.

Another example is emergency applications in response to natural disasters (e.g. floods and earthquakes) and security crises (e.g. terrorist attacks and riots). Resources provided through micro-clouds can be used to set up instant stations to relay safety information, locate victims, coordinate communication between rescue and security units, and provide alternative connectivity means (e.g. mounted on UAVs) in case long-haul access is lost.

## IV. Feasibility Experiments

We now focus on assessing the feasibility of using micro-cloud computing capabilities to deliver fog services, particularly to operate isolated execution environments at the edge. We zoom in on the Raspberry Pi (rPi) as the prominent micro-cloud component device due to its wide availability and affordability. The rPi generations used are summarised in TABLE 1, all equipped with a Wintec 16GB Class-10 microSD card.

TABLE 1 - Raspberry Pi machine specifications

| Model | PCB Ver. | Processor | | Cache (kB) | | Memory (MB) | Power (mA) |
|---|---|---|---|---|---|---|---|
| | | #Cores | Clock rate | L1 | L2 | | |
| rPi1B | 1.0 | 1 | 700 MHz | 16 | 128 | 256 | 300 |
| rPi1B | 2.0 | 1 | 700 MHz | 16 | 128 | 512 | 700 |
| rPi1B+ | 1.2 | 1 | 700 MHz | 16 | 128 | 512 | 600 |
| rPi2B | 1.1 | 4 | 900 MHz | 32 | 512 | 1024 | 800 |
| rPi3B | 1.2 | 4 | 1.2 GHz | 32 | 512 | 1024 | 800 |

We choose to investigate the ability to run different customer facing services (such as web caching, aggregation, pre-processing) in Docker containers as a realistic way of achieving isolation in micro-cloud devices. It also provides migration readiness in order to cater to changes in user requirements. We identify 4 key metrics to assess the ability of an rPi-based micro-cloud to handle isolated services at the edge: serving latency, hosting capacity, I/O overhead, and startup latency. Our rPis run HypriotOS v0.7.0 (unless otherwise stated), a Debian-based Linux distribution geared specifically towards running Docker over ARM processors.

### *A. Serving Latency*

Our first experiment investigates the responsiveness of application servers deployed on micro-clouds and their ability to serve a large number of requests. We deploy Apache httpd serving a webpage with minimal HTML and one image (≈100kB) in 2 settings: *native*, i.e. over Linux, and *Dockerized*, i.e. running inside a container. We then use the benchmarking tool Apache ab to stress test the servers with varying number of concurrent clients between 50 and 250, reaching a total of 1,000 clients per test. For these experiments representing an edge deployment, the rPi servers were within 2 hops and ≈20ms from the simulated clients. The results are plotted in FIGURE 2.

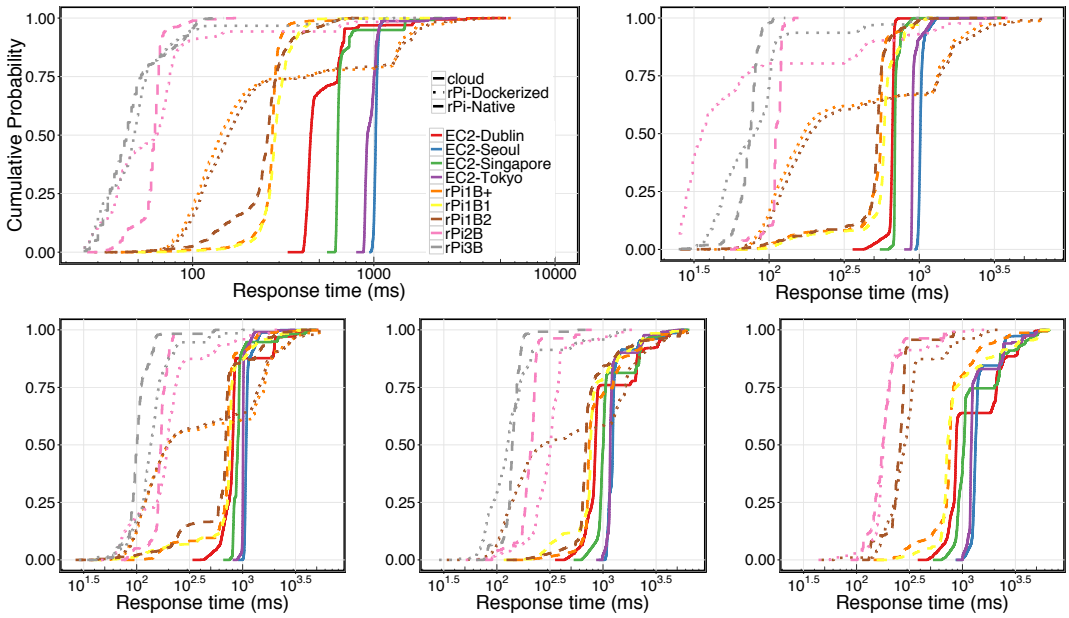



FIGURE 2 – CDFs of page retrieval times for different server setups with varying number of concurrent users: (top row from left) 50, 100, (bottom row) 150, 200, 250 users. rPis greatly reduce latency for users far from data centers.

We also deployed a similar server setup (only native httpd) on Amazon EC2 and ran the ab tests from Lahore, Pakistan. These cloud servers were located in the following locations offered by Amazon: Dublin, Seoul, Singapore, and Tokyo. The results are also presented in FIGURE 2 (keys starting with "EC2-").

We draw the main observations from this set of results. First, the results confirm that *a classic cloud deployment is not ideal for all scenarios*, especially for end users in locations like Pakistan where data centers (even Asia-Pacific ones) are at a considerable network distance. In such situations, micro-clouds provide a suitable substitute for applications requiring low latency. Second, most rPis are very capable of handling a significant number of web requests. Despite their limited computational capability, using them in such locations *improves latency by at least an order of magnitude*. However, as the number of concurrent users becomes significantly high, a hybrid deployment leveraging both remote data centers and micro-clouds becomes a more viable option as the micro-cloud computational limitations start to show. Third, running the servers within Docker introduces a notable amount of overhead attributed to isolation. In the case of the Pi1B1, the oldest of the rPis, the native server was able to serve 250 concurrent users while the Dockerized server was unable to serve more than 30. As a final comment, we note that successive rPi generations are becoming increasingly able to withstand additional load.

B. Hosting Capability

In our second experiment we explore the limits of hosting multiple Docker containers on an rPi. Although it is technically possible for a rPi to simultaneously run hundreds of Dockerized httpd servers, we found that containers

become practically unusable because the Docker daemon eventually starts pushing newly created containers to virtual memory. Consequently, containers require a significant delay before becoming usable as memory paging increases. Additionally, the rPi overall becomes rather unresponsive. Therefore, instead of examining the technical maximum container count, we instead set out to find out the real limit beyond which additional Docker containers become excessive.

We also examine how different production-grade services behave in these terms, rather than looking at a single example service. If micro-clouds are to become used as an everyday cloud analogue they will be required to handle a wide variety of services including load balancers, web servers, caches, messages queues, and databases. Our evaluation therefore includes a look into how different services operate on micro-clouds, covering the following representative applications that were all packaged for running on HypriotOS:

- CrateDB (database)
- httpd (web server)
- Jenkins (automation, integration)
- Ruby (programming runtime)
- ZooKeeper (coordination, synchronization)

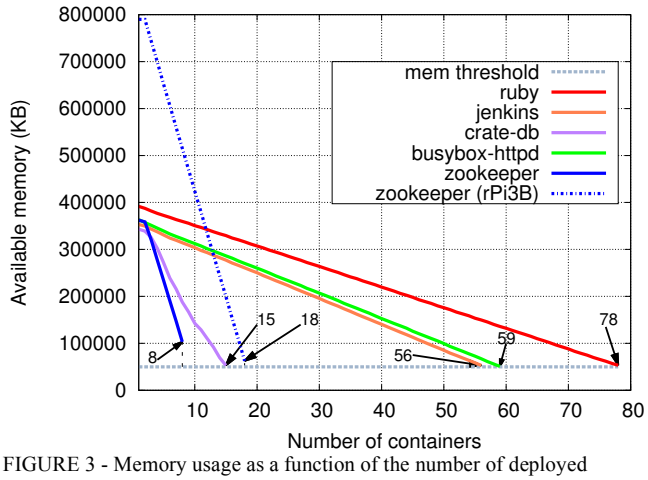



FIGURE 3 - Memory usage as a function of the number of deployed containers (10 runs). An rPi is capable of hosting a significant number of responsive containers, varying according to application.

We systematically deploy one Docker container after another, monitoring memory utilization and ending the experiment when a threshold of minimum available memory (50MB) is reached..The threshold preserves a certain amount of memory to the applications and the OS.

The experiments (FIGURE 3) were performed on rPi1B2 and rPi1B+ where available user space memory starts at ~380MB. As applications have different memory requirements, the maximum number of containers varies across

deployments: from 8 in the case of ZooKeeper to 78 for Ruby. We also note that ZooKeeper deployments never reached the threshold, stopping well before this as memory runs out before reaching the threshold. We repeated the experiment using rPi3B (running v0.8.0) where available memory was around 800MB. The rPi3B (dashed blue line) was able to deploy more than twice as many ZooKeeper containers as the older models.

Contrary to expectation, the rPi's secondary storage did not have a significant impact. For instance, deploying 60 containers of the httpd server requires only 8.3kB of disk storage. However, the experiments revealed that main memory is a significant bottleneck.

### *C. I/O Overhead*

Our third experiment dives into the performance that the rPi architecture delivers to the hosted applications. Besides CPU speed, one of the major differences in rPi hardware architecture is the physical memory design. Memory access also represents one of the major costs involved in a range of Internet services, from databases to serving resources and processing data. We therefore examine the relative cost of reading from and writing to different kinds of memory with our various rPi models, compared to a cloud server. To do this we wrote a program which reads and writes increasingly large amounts of data to secondary storage, and also writes increasingly large amounts of data in RAM (we consider main memory reads and writes to be symmetrical).

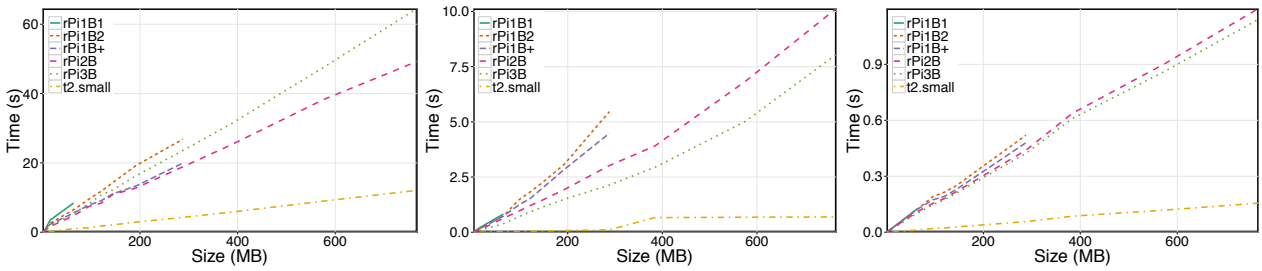



FIGURE 4 - Disk and memory access times (20 runs). Writing to disk on rPis is expensive, but progressively getting better for newer models.

The results, depicted in FIGURE 4, demonstrate the following. First, cloud I/O speeds are considerably faster across all types of memory access; this is expected due to their generally higher hardware specification (i.e. CPU speed, cache architectures, system bus speed and main memory latency). However, successive rPi models provide incremental improvements in I/O speeds. Second, writing to disk contains a far higher relative penalty across all rPi models than in the server case (compared to disk read or memory access speeds). We assume that the relative additional disk write latency in rPi systems is caused by the relatively slow write speeds on flash memory used in SD cards [9]. Software deployments that predominantly use disk reads and memory access, avoiding disk writes, may therefore be even more valuable for efficiency on these devices than on cloud-based servers — otherwise the network latency gained through geographical proximity may be eclipsed by slower disk access. Of the applications we tested, thus, httpd, Ruby and ZooKeeper are most likely to be suited to the rPi environment. Emerging trends in

system design such as in-memory databases may also be particularly useful, though obviously main memory is capacity-limited. In the wider research picture, because disk access of applications may not be predictable ahead of time, work on adaptive systems may further help to gain the best balance between traditional- and micro-cloud deployments, selecting the optimal placement of a server based on real-time observations.

### D. Startup Latency

Our final experiment measures booting time, which is of importance for deployments where electricity shortage is a chronic problem.

We include a baseline of a t2.small EC2 instance running Ubuntu (*without* network delay). This takes 27.58s to boot (all figures are mean of 20 runs). rPis take less time: 21.97s, 22.02s, and 22.40s for rPi1B1, rPi1B2, and rPi1B+ respectively. More recent rPi models undercut EC2 by over 10 seconds, a 40% margin: 16.74s and 16.06s for rPi2B and rPi3B respectively.

Starting Docker on rPi takes considerably more time, though: 5.89s, 5.90s, 5.44s, 3.05s, 4.86s; but only 0.21 on the EC2 instance. However, even with this additional delay, one would have a running Docker container on rPi2B or rPi3B up to 8 seconds before an EC2 instance is ready. Note that these are pure OS and hypervisor latencies without accounting for network latency, which would tip the advantage in rPi's favour even further.

### *E. Migration Policy*

Based on the findings on serving and startup latencies, we could develop a simple policy to invoke migration of an application from the cloud to a microcloud as follows. Let $t$ be the latency threshold whereby migration is triggered if $t>0$.

$$t = x_c - (x_m + m) = (s_c + l_c) - (s_m + l_m) - m$$

where $x_c$ is the total latency of the cloud application and is made up of the startup latency $s_c$ and serving latency $l_c$; $x_m$, $s_m$, and $l_m$ are the equivalent for the candidate microcloud; and $m$ is the latency to transfer any required state. Migration would only make sense if the application lifetime is expected to be longer than the cost of migration, i.e. $x_m+m$. All variables but $m$ can be obtained as the medians of our experiment results above. There are different approaches to estimating $m$, but this is not our focus here.

## V. State of the Art

Having explored feasibility, we now consider the state of the art in micro-cloud implementations and promising future directions. Work in this domain revolves around 3 main axes: hardware, resource management, and programming abstractions.

### *A. Hardware*

Small single-board computers, the building blocks of micro-clouds, are advancing all the time with better chips, additional modules, lower power requirements, and smaller size. The major challenge here is to assemble such

devices to build micro-clouds whilst minimizing internal power and network wiring to reduce assembly cost. Several commercial and research ventures have already started work on this using different strategies. Examples include: PiFace Rack, RuggedPOD, Iridis-Pi [10], Pi Beowulf cluster [11], BitScope, PicoCluster, and Grape Cluster.

### *B. Resource Management*

Ongoing efforts to design OS and orchestration tools suitable for this scale of computers are revolving around container technologies and the microservices architectural philosophy. HypriotOS is a leading effort here, yet it is a general purpose OS. There is room for a leaner OS, and for operating isolated application stacks using technologies other than Docker, e.g. Unikernels [12] and ContainerX.

### *C. Programming Models*

A few works undertook building common tools and vocabulary to simplify setting up and operating fog systems. Mobile Fog [13] defines a specification for location-aware applications. Zhang et al [14] introduce a data-centric abstraction API based around distributed logs accessible through names not locations. The Holon architecture [15] is a more generic, conceptual framework to support the composition of systems-of-systems.

More solutions are required in this direction to support designing applications that are readily divisible between multiple deployment infrastructures. For instance, an application's presentation layer could reside on several micro-cloud instances close to the users, with a shared data tier managed in a cloud data center.

Most commercial contributions focus on integration frameworks for machine-to-machine communications in-between colocated devices and cloud servers. This simplifies the development and maintenance of solutions such as home automation, personal healthcare, and smart traffic systems. Examples include Arkessa, Axeda, Lyric, Resin, SmartBear, Thingsquare, and Withings.

## VI. CONCLUDING REMARKS

Our results are the first to empirically evaluate the suitability and performance trade-offs of micro-clouds. We demonstrate them to be adequate hosting environments for concurrent edge web services, able to serve a large number of requests while preserving their responsiveness. Moreover, they boot fairly quickly, making them suitable for dynamic deployments where power is intermittent. The only limitation relates to I/O heavy applications as writing to disk is extremely expensive, but is progressively getting better.

These benefits apply to applications that are either resource-rich (e.g. smart city IoT deployment) or resource-poor (e.g. remote community). Our network latency results demonstrate significant potential for moving services closer to the consumer using cheap, micro-service deployments, as long as the volume of co-hosted services and the type of service are carefully considered.

## ACKNOWLEDGMENT

This work was partially funded by CHIST-ERA under UK EPSRC grant reference EP/M015734/1 and Swiss SNSF grant reference 155249; and also by EPSRC grant reference EP/M029603/1.

## REFERENCES

[1] L. M. Vaquero and L. Rodero-Merino, "Finding your way in the fog: Towards a comprehensive definition of fog computing," *SIGCOMM Computer Communication Review*, vol. 44, no. 5, pp. 27– 32, ACM, Oct 2014.

[2] M. Satyanarayanan, P. Bahl, R. Caceres, and N. Davies, "The case for VM-based cloudlets in mobile computing," *Pervasive Computing*, vol. 8, no. 4, pp. 14–23, IEEE, Oct 2009.

[3] J. Crowcroft, A. Madhavapeddy, M. Schwarzkopf, T. Hong, and R. Mortier, "Unclouded vision," *Conference on Distributed Computing and Networking*. Springer, 2011, pp. 29–40.

[4] Nordpil, "World database of large urban areas, 1950-2050," https://nordpil.com/resources/world-database-of-large-cities/ Nordpil & the UN Population Division, Mar 2010.

[5] Y. Elkhatib, "Building cloud applications for challenged networks," *Embracing Global Computing in Emerging Economies*. Springer, Nov 2015, pp. 1–10.

[6] F. Bonomi, R. Milito, J. Zhu, and S. Addepalli, "Fog computing and its role in the internet of things," *Workshop on Mobile Cloud Computing*. ACM, 2012, pp. 13–16.

[7] P. G. Lopez, A. Montresor, D. Epema, A. Datta, T. Higashino, A. Iamnitchi, M. Barcellos, P. Felber, and E. Rivière, "Edge-centric computing: Vision and challenges," *SIGCOMM Computer Communication Review*, vol. 5, no. 45, ACM, Oct 2015.

[8] Weisong Shi and Schahram Dustdar, "The Promise of Edge Computing," *Computer*, vol. 49, no. 5, IEEE, 2016, pp. 78-81.

[9] F. Chen, D. A. Koufaty, and X. Zhang, "Understanding intrinsic characteristics and system implications of flash memory based solid state drives," *SIGMETRICS Performance Evaluation Review*, vol. 37, no. 1, pp. 181–192, ACM, Jun 2009.

[10] S. J. Cox, J. T. Cox, R. P. Boardman, S. J. Johnston, M. Scott, and N. S. O'Brien, "Iridis-Pi: A low-cost, compact demonstration cluster," *Cluster Computing*, vol. 17, no. 2, pp. 349–358, 2013.

[11] J. Kiepert, "Creating a Raspberry Pi-based Beowulf cluster," Boise State University, Tech. Rep., 2013.

[12] A. Madhavapeddy, R. Mortier, C. Rotsos, D. Scott, B. Singh, T. Gazagnaire, S. Smith, S. Hand, and J. Crowcroft, "Unikernels: Library operating systems for the cloud," *SIGPLAN Notices*, vol. 48, no. 4. ACM, 2013, pp. 461–472.

[13] K. Hong, D. Lillethun, U. Ramachandran, B. Ottenwälder, and B. Koldehofe, "Mobile fog: A programming model for large-scale applications on the internet of things," *Workshop on Mobile Cloud Computing*. ACM, 2013, pp. 15–20.

[14] B. Zhang, N. Mor, J. Kolb, D. S. Chan, K. Lutz, E. Allman, J. Wawrzynek, E. Lee, and J. Kubiatowicz, "The cloud is not enough: Saving IoT from the cloud," *Workshop on Hot Topics in Cloud Computing*. USENIX, Jul 2015.

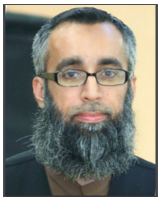
[15] G. Blair, Y.-D. Bromberg, G. Coulson, Y. Elkhatib, L. Réveillère, H. B. Ribeiro, E. Rivière, and F. Taïani, "Holons: Towards a systematic approach to composing systems of systems," *Workshop on Adaptive and Reflective Middleware*. ACM, 2015, pp. 5:1–5:6.

**Yehia Elkhatib** is an Assistant Professor at Lancaster University, UK working on traversing infrastructural boundaries, composing intent-driven network architectures, and measuring networked systems and protocols.
<y.elkhatib@lancaster.ac.uk>

**Barry Porter** is an Assistant Professor at Lancaster University, UK working on emergent, self-aware and self-adaptive software systems in a range of application areas including IoT.
<b.f.porter@lancaster.ac.uk>

**Heverson B. Ribeiro** is a post-doctoral researcher at the University of Neuchâtel, Switzerland. His interests lie in large-scale distributed storage systems and algorithms, edge computing and peer-to-peer systems.
<heverson.ribeiro@unine.ch>

**Mohamed Faten Zhani** is an Assistant Professor at École de Technologie Supérieure (ÉTS), University of Quebec, Canada. His research interests include cloud computing, virtualization, Big Data analytics, software defined networks and resource management in large-scale distributed systems.
<mfzhani@etsmtl.ca>



**Junaid Qadir** is an Associate Professor at the Information Technology University (ITU) Punjab in Lahore, Pakistan. He is the Director of the IHSAN Lab at ITU that focuses on systems and networking research as well as deploying ICT for development.
<junaid.qadir@itu.edu.pk>

**Etienne Rivière** is a Lecturer at the University of Neuchâtel, Switzerland. His research interests lie in large-scale distributed systems and concurrent systems.
<etienne.riviere@unine.ch>