Edge-as-a-Service: Towards Distributed Cloud Architectures

Varghese, B., Wang, N., Li, J., & Nikolopoulos, D. S. (2017). Edge-as-a-Service: Towards Distributed Cloud Architectures. In International Conference on Parallel Computing (pp. 784-793). (Advances in Parallel Computing). https://doi.org/10.3233/978-1-61499-843-3-784

Published in:
International Conference on Parallel Computing

Document Version:
Peer reviewed version

Queen's University Belfast - Research Portal:
Link to publication record in Queen's University Belfast Research Portal

October 2017

Edge-as-a-Service: Towards Distributed Cloud Architectures

Blesson VARGHESE [a,1], Nan WANG [a], Jianyu LI [a] and Dimitrios S. NIKOLOPOULOS [a]

[a] *School of Electronics, Electrical Engineering and Computer Science*
Queen's University Belfast, UK

Abstract. We present an Edge-as-a-Service (EaaS) platform for realising distributed cloud architectures and integrating the edge of the network in the computing ecosystem. The EaaS platform is underpinned by (i) a lightweight discovery protocol that identifies edge nodes and make them publicly accessible in a computing environment, and (ii) a scalable resource provisioning mechanism for offloading workloads from the cloud on to the edge for servicing multiple user requests. We validate the feasibility of EaaS on an online game use-case to highlight the improvement in the QoS of the application hosted on our cloud-edge platform. On this platform we demonstrate (i) low overheads of less than 6%, (ii) reduced data traffic to the cloud by up to 95% and (iii) minimised application latency between 40%-60%.

Keywords. edge computing, distributed cloud, resource discovery, Edge-as-a-Service

Introduction

The upcoming Internet-of-Things paradigm motivates the need for developing distributed cloud architectures to meet future computing challenges [1]. Distributed cloud architectures tap into spare computing that may be available at the edge of the network (for example on local routers, mobile base stations and switches [2]). However, this is challenging and not an easy task. Alternatively, it is proposed that low cost and low power computing nodes, such as Raspberry Pis can be placed one hop away from user devices, sometimes referred to as micro clouds. These nodes can host servers offloaded from cloud servers to service a set of user devices for improving the overall QoS [3].

The edge of the network will need to be integrated into the computing ecosystem for realising a distributed cloud architecture. For example, micro clouds will need to be accessible to applications and application owners if a server is to be deployed on them. There are public services to enable the selection of cloud resources and deployment of applications, such as the Amazon Web Services for the Elastic Compute Cloud. However, there are no such services to enable the selection and deployment of applications on edge nodes. This paper addresses the problem by proposing and developing the first *'Edge-as-a-Service'* (EaaS) platform to makes edge nodes publicly available.

[1] Corresponding Author. E-mail: varghese@qub.ac.uk; Web: www.blessonv.com

The EaaS platform is built on a discovery protocol. A collection of homogeneous edge nodes are identified and made publicly accessible via a controller. We used Raspberry Pis[2] as edge nodes and implemented management algorithms to make edge node services accessible in the platform. The feasibility of the EaaS platform was explored in two ways. Firstly, by experimentally investigating the overheads in the platform. The key observation is that our platform has an overhead of up to 6%. Secondly, by developing a real use-case, which is a location-aware online game that make use of distributed cloud architectures. The key result is that the application latency is reduced between 40%-60% for up to 1024 users and the edge node processes nearly 95% data.

1. Edge-as-a-Service (EaaS) Platform

The EaaS platform operates as a three tier architecture as follows. The top tier is the *cloud layer* which hosts application servers. A centralised cloud architecture will be typically a two-tier model in which user devices connect to the application server.

The bottom layer is the *device layer* comprising user devices, such as smartphones, wearables and gadgets that connect to cloud application servers. Typically in a centralised cloud architecture, devices connect to the application servers through traffic routing nodes. However, to include computing services offered by edge nodes, user devices will need to connect to edge nodes through the cloud server.

The middle tier is the *edge node layer* in which edge node(s) are made available on-demand to support a collection of user devices that may be close to the node(s). For example, when users start an application, a connection is firstly established with the cloud application servers. Then an application server (either a clone or a partitioned server) may be deployed on an edge node.

The EaaS platform is underpinned by mechanisms for (i) discovering edge nodes, and (ii) provisioning resources on edge nodes for workloads.

1.1. Component View

The EaaS platform that operates in the edge node layer requires a master node (located either in the edge node layer or elsewhere) and a collection of edge nodes. The master node executes an EaaS controller that connects to a collection of edge nodes. Each edge node executes an EaaS manager. The Master Node Controller comprises the following six modules (refer Figure 1).

1) The *Master Node Manager* is the interface between the user input from the Web Application and the Master Controller. This module interacts with the Monitor and the Key Generator. Information to isolate users is maintained by the User Database on the Master Node. The Edge Database contains information on all edge nodes that have been discovered and have installed an Edge Manager on the edge node.

2) The *Monitor* periodically obtains key metrics, such as the CPU and memory utilisation from each edge node. The frequency of obtaining this information can be set by the Administrator. However, currently this module is centralised and we anticipate that it will be less scalable. Alternate distributed monitoring mechanisms will need to be implemented to make the EaaS platform more scalable.

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

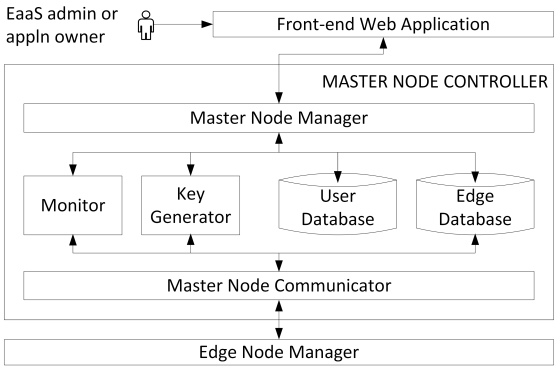


Figure 1. Component view of the EaaS platform

3) The *Key Generator* enhances security of the EaaS platform by generating a public and private key for containers. Users are provided can download a private key via the Web Application when a container is launched for the first time.

4) The *User Database* maintains the users of the EaaS system by managing usernames and passwords. The current active users on the edge nodes and their time of activity are logged in this database.

5) The *Edge Database* stores the information of all discovered edge node, such as its public IP, open port number and coordinator information and of containers on the edge node, namely its name, state and public IP.

6) The *Master Node Communicator* is the interface between the Controller and the Edge Node and communicates with edge node manager. Commands to launch, start, stop and terminate containers on the edge nodes are assembled in this module. The results of executing the commands on the edge node are obtained from the Edge Node Manager and used to update the database. Additionally, the metric information obtained from Monitoring is obtained by the Monitor via this module.

The *Edge Node Manager* is installed on the edge node during discovery and executes commands obtained from Master Node Communicator and provides the output of the execution back to the Communicator.

The *Web Application* provides a user interface for accessing the EaaS platform. The users may be administrators of the EaaS platform or application owners who require access to an edge node. The administrator interface provides global information on a variety of metrics, such as CPU utilisation and memory utilisation of containers that are deployed by users on the edge node. The administrator can start, stop or terminate containers that are running. The application owner interface provides information relevant to the containers that a user has deployed on an edge node. The users can choose to launch an application container (for example, Docker) or an Operating System container (for example, LXD) on the edge node.

1.2. Discovery and Provisioning Services

Table 1 shows the mathematical notation employed for describing the discovery protocol and the services offered by the EaaS platform. The parameters described in the table are provided either by the Master Node Controller, the Edge Node or the Edge User. The

Table 1. Notation used in the proposed EaaS platform

Parameter	Description	Source
$Prt_{controller}$	Port used on controller for communication with edge nodes	Master Node Controller
$IP_{controller}$	IP address of the controller	
E	A set of n edge nodes offered as a service, $e_i \in E, i = 1,...,n$, where e_i = $[IP_{e_i}]$	
C	A set of m containers launched on E, $c_i \in C, i = 1,...,m$, where c_i = $[IP_{c_i}, status_{c_i}]$	
IP_{e_i}	IP address of edge node e_i	Edge Node
Prt_e	Port used on edge nodes for communication with controller	
$offer_i$	A list of $[e_i, IP_{e_i}, Prt_e]$ to offer services on edge node e_i	
$service$	Flag on the availability of the services on an edge node	
key_{c_i}	Private key generated for accessing c_i	
$result_i$	Results generated by an application in c_i	
IP_{c_i}	IP address of container c_i	
$status_{c_i}$	Status of container c_i	
$conType$	Flag for whether an OS or application container is deployed	Edge Service User
$action_i$	Actions (launch, terminate, start or stop) on a container c_i	
$request_i$	A list of $[conType, e_i, action_i, c_i]$ to request services on edge node e_i	

discovery protocol identifies edge nodes that communicate with the master node. Each edge node installs a simple Manager initially in the discovery.

Algorithm 1 presents the discovery protocol in which edge nodes are brought in a single environment of the EaaS platform. The algorithm shows the activities on the Master Node Controller and the Edge Node Manager. The port to communicate with the Master Node Controller is known to the Edge Node Manager (Line 1). A list of all edge nodes and their services are maintained by the Controller (Lines 2 and 3) because the Edge Node Manager updates the Controller when it can offer a service (Lines 5 and 6).

Procedure 1: Discovery protocol

on the **Master Node Controller**
Data: $Prt_{controller}, E, offer_i$
1 listen $Prt_{controller}$;
2 **if** $offer_i$ **then**
3 | update E with e_i;
4 **end**
;
on the **Edge Node Manager**
Data: $IP_{controller}, Prt_{controller}, service, offer_i$
5 **if** $service == True$ **then**
6 | send $offer_i$ to controller using $IP_{controller}, Prt_{controller}$;
7 **end**

Procedure 2, Procedure 3 and Procedure 4 presents how different services are offered in the EaaS platform and the corresponding functions on the Front-end, Master Node Controller and Edge Node Manager respectively.

On the Front-end, a user requests services to the Controller for either launching, terminating, starting or stopping containers on an edge node (Line 1). If the request is to start, stop or terminate containers then these actions are performed. However, if an OS container is to be launched then the user is provided access to the container via a private key (Lines 3-6). If an application container is to be launched, then it is executed on the edge node and the results are obtained (Line 9).

Procedure 2: Edge service procedure on the **Front-End**

```
   Data: Prt_controller, IP_controller, request_i, key_ci, result_i
 1 send request_i to controller using IP_controller, Prt_controller;
 2 if action == 'launch' then
 3 |   if conType == 'os' then
 4 |   |   if key_ci, IP_ci then
 5 |   |   |   download key_ci;
 6 |   |   |   access to IP_ci;
 7 |   |   end
 8 |   else
 9 |   |   receive result_i;
10 |   end
11 end
```

On the Master Node Controller, when a request is received from the Front-End it is passed on to the Edge Node Manager (Line 1). If the request is to launch an OS container then the private key generated by the Controller is provided to the user (Lines 3-4). The list of containers on each edge node is updated (Line 6). Alternatively, if the request from the Front-end was to launch an application container, then the results obtained from the Edge Node Manager are displayed on the Front-end (Line 9).

Procedure 3: Edge service procedure on the **Master Node Controller**

```
 1 request_i, C, key_ci, IP_ci, status_ci, result_i send request_i to e_i
 2 if conType == 'os' then
 3 |   if key_ci, IP_ci then
 4 |   |   send key_ci, IP_ci to Front-end
 5 |   end
 6 |   update C with status_ci, IP_ci
 7 else
 8 |   if result_i then
 9 |   |   send result_i to Front-end
10 |   end
11 end
```

On the Edge Node Manager, LXD commands are launched for an OS container and Docker containers are used for application containers. If the request is to launch a container, then Lines 4-7 are executed. If an application container is to be launched then

lines 9-10 are executed. Similarly for start (Lines 14-16), for stop (Lines 19-20) and for terminate (Lines 23-24) are executed.

Procedure 4: Edge service procedure on the **Edge Node Manager**

```
1  request_i switch action do
2      case launch do
3          if conType == 'os' then
4              launch LXD container c_i
5              generate key_{c_i}
6              configure IP_{c_i}
7              send key_{c_i}, IP_{c_i}, status_{c_i} to controller
8          else
9              launch Docker container c_i
10             send result_i to controller
11         end
12     end
13     case start do
14         start LXD container c_i
15         configure IP_{c_i}
16         send IP_{c_i}, status_{c_i} to controller
17     end
18     case stop do
19         stop LXD container c_i
20         send status_{c_i} to controller
21     end
22     case terminate do
23         terminate LXD container c_i
24         send status_{c_i} to controller
25     end
26 end
```

2. Use-case

The feasibility of EaaS is demonstrated on an open-sourced version of a location-aware online game similar to PokéMon Go, named iPokeMon[3]. The game features a virtual reality environment that can be played on a variety of devices, such as smartphones and tablets. The user locates, captures, battles and trains virtual reality creatures, named Pokémons, through the GPS capability of the device. The Pokémons are geographically distributed and a user aims to build a high value profile among their peers. The users may choose to walk or jog through a city to collect Pokémons.

The current execution model uses a centralised cloud architecture, such that the game server is hosted on the public cloud and the users connect to the server. The server

[3] `https://github.com/Kjuly/iPokeMon`

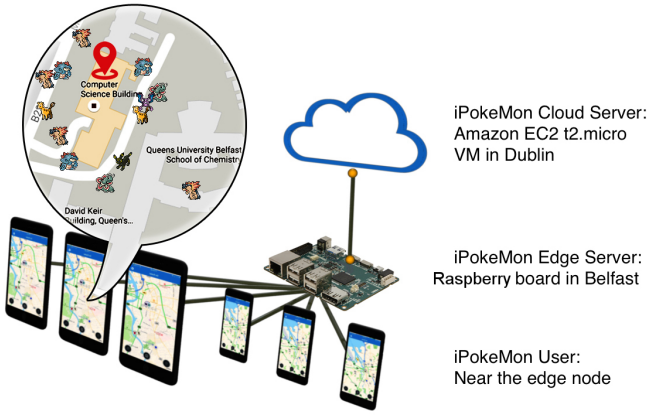


Figure 2. Distributed iPokeMon game using the EaaS plaform.

updates the user position and a global view of each user and the Pokémons is maintained by the server. For example, if Amazon Elastic Compute Cloud (EC2) servers are employed, then the game may be hosted in an EC2 data center and a user in Belfast communicates with the game server. The original game server is known to have crashed multiple times during its launch due to severe activities which were not catered for[4].

We used the EaaS platform to employ a distributed cloud architecture for executing the iPokeMon game as shown in Figure 2. The game server is hosted in the Amazon EC2 Dublin data center on a t2.micro instance. A partitioned game server is deployed using an LXD container on an edge node which executes the edge node manager of the EaaS platform. The server was manually partitioned (the cloud server maintained a global view of the Pokémons, where as the edge node server had a local view of the users connected to the edge server). The edge node periodically updated the global view of the cloud server. User devices connect to the edge node instead of the cloud server to service requests (process GPS coordinates and update personal profile).

3. Evaluation

In this section, we evaluate the proposed EaaS platform by measuring the overheads associated with deploying workloads on edge nodes. Before this we consider the test-bed we have developed for the EaaS platform.

3.1. Platform

The edge nodes used are a set of three Raspberry Pi 2 boards. Each board comprises a 900MHz quad-core ARM Cortex-A7 CPU and 1 GB RAM. An Ubuntu MATE 16.04.2 LTS image is copied to a micro SD card that is used by the Raspberry Pi as the operating

[4] `http://www.forbes.com/sites/davidthier/2016/07/07/pokemon-go-servers-seem-to-be-struggling/\#588a88b64958`

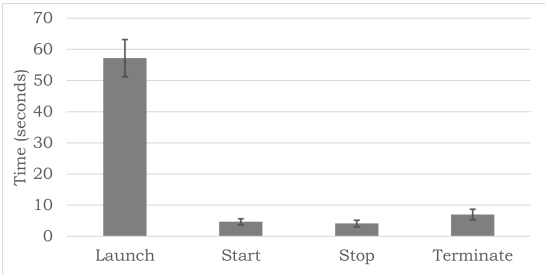


Figure 3. Overall round trip latency

system. LXD containers are employed for launching workloads on the Raspberry Pi. The Operating System (OS) of the containers that are launched is Alpine Linux.This OS creates a lean container (other OS' have additional packages installed). The Edge Node Manager is executed locally on the Pis. The edge nodes are connected via a switch to an EaaS master node. The master node is an Intel i7-4720HQ CPU @ 2.60GHz 2.59GHz and 8 GB RAM system.

3.2. Measuring Overheads

The average overhead involved in using the EaaS platform on three edge nodes of our experimental test-bed was considered.

Our experiments indicated that the overheads remain nearly constant when 50 containers are used (these results are exhaustive and not in the scope of this paper). The time taken to launch a container is under 1 minute on each edge node, even when 50 containers are launched. A container can be started and stopped in under 5 seconds and terminated in nearly 7 seconds. The results show that the EaaS platform is scalable up to 50 containers. Experiments were not pursued beyond 50 containers since the resources are exhausted on the edge node by the container.

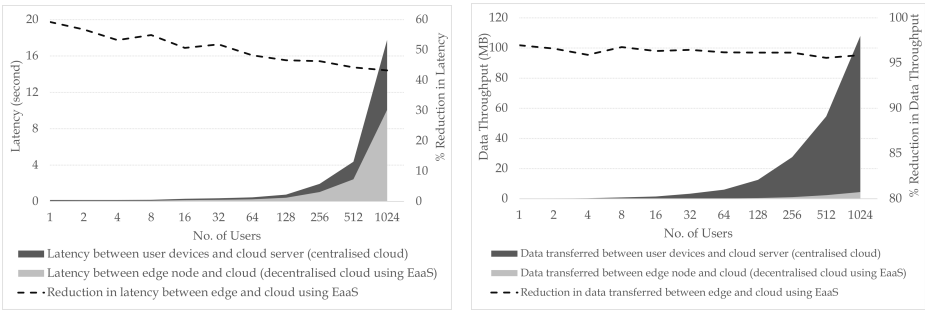
Figure 3 shows the average overall round trip latency (when the user initiates the operation on the front-end, which is then passed on to the master node and after which the edge node manager executes the operation on the edge node) for launching, starting, stopping and terminating containers and the standard deviation on the EaaS platform. The average overhead that is added by the front-end and master node in the EaaS platform for managing the edge nodes is shown in Table 2. We do not account for the time taken for launching, starting, stopping and terminating a container on an edge node in calculating the overhead since it cannot be eliminated if containers are used on the edge node. An overall overhead of nearly 6% is added by the EaaS platform for launching the container. This is higher than the overheads for starting, stopping and terminating containers because the master node deals with generating the key on the edge node.

3.3. EaaS for the Use-case

Figure 4a shows the average latency experienced by the user, which is measured by round trip latency from when the user device generates a request while playing the game that needs to be serviced by a cloud server (this includes the computation time on the server). The response time is noted over a five minute time period for varying number of

Table 2. Percentage overhead in using the front-end and master node of the EaaS platform

Service	Communication Overhead (%)		
	Between front-end and EaaS master node	Between EaaS master node and an edge node	Overall
Launch	0.54	5.36	5.90
Start	0.06	0.91	0.97
Stop	0.07	1.06	1.13
Terminate	0.04	0.64	0.68



(a) Latency of iPokeMon game users when using a server located on the cloud and on an edge node.

(b) Percentage reduction in the data traffic between edge nodes and the cloud.

Figure 4. Latency and reduction in data traffic using EaaS

users. Using EaaS for a decentralised cloud architecture, it is noted that on an average the latency is reduced between 40%-60% for up to 1024 users playing the game.

Figure 4b presents the amount of data that is transferred during a fifteen minute time period. As expected with increasing number of users the data transferred increases. However, we observe that using EaaS the data transferred between the edge node and the cloud is significantly reduced, yielding an average of over 95% reduction.

4. Related Work

Different architectures have been proposed at the data center level for achieving distributed clouds. For example, hybrid [4], federated [5] and ad hoc [6] clouds.

More recently to address the connectivity and data challenges that are expected from the IoT paradigm cloud architectures extending to resources outside data centers have been proposed [7]. They decentralise the concentration of compute resources away from data centers towards the edge of the network [8]. This helps in bringing computing closer to the source of data thereby reducing traffic beyond the first hop in the network.

The deployment of cloudlets, which are dedicated nodes at the edge of the network can service requests of mobile devices in a distributed manner [9,10]. Hence, mobile edge computing has emerged [11].Similarly, micro clouds, which are miniature data centers are reported in literature [12,13].This has resulted in edge and fog computing [14,15].

Despite the advent of these architectures it is challenging to integrate the edge of the network in a cloud architecture [16]. The key problem that needs to be addressed is

making edge nodes publicly accessible. This will allow application servers running on the cloud to offload workloads onto edge nodes to service requests originating from a given location. To this end, edge nodes will need to be firstly discovered and then be brought under a single environment that makes them visible. This paper aims to address this gap by developing an Edge-as-a-Service platform.

5. Conclusions

The EaaS we propose and develop aims to integrate the edge of the network in the computing ecosystem. This paper presents the first steps we have taken in realising distributed cloud architectures for meeting the challenges of emerging distributed workloads.

References

[1] B. Varghese and R. Buyya. Next Generation Cloud Computing: New Trends and Research Directions. *Future Generation Computer Systems*, 2017.

[2] C. Meurisch, A. Seeliger, B. Schmidt, I. Schweizer, F. Kaup, and M. Mühlhäuser. Upgrading Wireless Home Routers for Enabling Large-scale Deployment of Cloudlets. In *Mobile Computing, Applications, and Services*, pages 12–29. 2015.

[3] N. Wang, B. Varghese, M. Matthaiou, and D. S. Nikolopoulos. ENORM: A Framework For Edge NOde Resource Management. *IEEE Transactions on Services Computing*, 2017.

[4] K. Zhang, X. Zhou, Y. Chen, X. Wang, and Y. Ruan. Sedic: Privacy-aware Data Intensive Computing on Hybrid Clouds. In *Proceedings of the 18th ACM Conference on Computer and Communications Security*, pages 515–526, 2011.

[5] B. Rochwerger, C. Vzquez, D. Breitgand, D. Hadas, M. Villari, P. Massonet, E. Levy, A. Galis, I. Llorente, R. S. Montero, Y. Wolfsthal, K. Nagin, L. Larsson, and F. Galn. *An Architecture for Federated Cloud Computing*, pages 391–411. John Wiley & Sons, Inc., 2011.

[6] G. A. McGilvary, A. Barker, and M. Atkinson. Ad Hoc Cloud Computing. In *Proceedings of the IEEE 8th International Conference on Cloud Computing*, pages 1063–1068, 2015.

[7] J. Gubbi, R. Buyya, S. Marusic, and M. Palaniswami. Internet of Things (IoT): A Vision, Architectural Elements, and Future Directions. *Future Generation Computer Systems*, 29(7):1645–1660, 2013.

[8] B. Varghese, N. Wang, S. Barbhuiya, P. Kilpatrick, and D. S. Nikolopoulos. Challenges and Opportunities in Edge Computing. In *IEEE International Conference on Smart Cloud*, pages 20–26. 2016.

[9] K. Gai, M. Qiu, H. Zhao, L. Tao, and Z. Zong. Dynamic Energy-aware Cloudlet-based Mobile Cloud Computing Model for Green Computing. *Journal of Network and Computer Applications*, 59(C):46–54, 2016.

[10] M. Satyanarayanan, P. Bahl, R. Caceres, and N. Davies. The Case for VM-Based Cloudlets in Mobile Computing. *IEEE Pervasive Computing*, 8(4):14–23, 2009.

[11] S. Wang, R. Urgaonkar, M. Zafer, T. He, K. Chan, and K. K. Leung. Dynamic Service Migration in Mobile Edge-Clouds. In *IFIP Networking Conference*, pages 1–9, 2015.

[12] F. P. Tso, D. R. White, S. Jouet, J. Singer, and D. P. Pezaros. The Glasgow Raspberry Pi Cloud: A Scale Model for Cloud Computing Infrastructures. In *33rd IEEE International Conference on Distributed Computing Systems Workshops*, pages 108–112, 2013.

[13] Y. Elkhatib, B. Porter, H. Ribeiro, M. Zhani, J. Qadir, and E. Rivire. On Using Micro-Clouds to Deliver the Fog. *IEEE Internet Computing*, 2016.

[14] S. Deng, L. Huang, J. Taheri, and A. Y. Zomaya. Computation Offloading for Service Workflow in Mobile Cloud Computing. *IEEE Transactions on Parallel & Distributed Systems*, 26(12):3317–3329, 2015.

[15] K. Bhardwaj, P. Agrawal, A. Gavrilovska, and K. Schwan. AppSachet: Distributed App Delivery from the Edge Cloud. In *7th International Conference Mobile Computing, Applications, and Services*, 2015.

[16] M. Villari, M. Fazio, S. Dustdar, O. Rana, and R. Ranjan. Osmotic Computing: A New Paradigm for Edge/Cloud Integration. *IEEE Cloud Computing*, 3(6):76–83, 2016.