Edge Computing Embedded Platform with Container Migration

Labhesh Deshpande, Kaikai Liu
Computer Engineering Department
San Jose State University (SJSU)
San Jose, CA, USA
Email: {labhesh.deshpande, kaikai.liu}@sjsu.edu

***Abstract*—In a world where the number of smart cities is growing exponentially, there is a myriad of IoT devices which are generating immense data, 24x7. Centralized cloud data centers responsible for handling this huge data are being rapidly replaced with distributed edge nodes which move the computation closer to the users to provide low latencies for real-time applications. The proposed enhancements capitalizes on this design and proposes an effective way to achieve fault tolerance in the system. The concept of docker container migration is used to provide a near-zero downtime system on a distributed edge cloud architecture. An intuitively simple and visually attractive dashboard design is also being presented in this paper to remotely access the edge cloud management services.**

Edge Computing, Fog Computing, Internet-of-Things, Multi-tenant Applications

I. Introduction

According to Cisco Global Cloud Index, by 2019, the data produced by people, machines, and devices will reach 500 zettabytes. It is difficult to provide such high bandwidth since the global data center IP traffic will only reach 10.4 zettabytes by the year 2019 [2]. There is a need of a system, that can proactively provide computing services for these IoT devices based on needs. Edge computing is a technology that allows the computation on micro-base station, present on the edge of the network and update the results over the network, which consequently, solves the problem of high network bandwidth requirement and as well response time. The edge computing platform uses docker containers to provide the required isolation of application by separating its execution from external stimuli and also achieve multi-tenancy by the reuse of the applications from different containers using virtualization. We are proposing a service migration architecture in the existing platform[3] to harness the capabilities of the edge nodes to their fullest and reduce the downtimes of the system as a whole by increasing Fault tolerance. Docker container migration is the addition that is being proposed to ensure that the system doesnt stop in a case where a container stops responding due to overloading or in a case it completely fails. Docker container migration basically checkpoints a live container, at one location, and then that container along with all its data is restored it at some another location. Users are echoed a connection interrupted message during the live migration and get their data back in the same state after the migration. This gives a much better experience to users as compared to receiving a complete loss of connection to the server and reconnecting to the server. Enhancing the Edge Computing Embedded platform using docker migration will take into account the same steps of the live migration and migrate live containers. After assessing the accessibility of the present architecture of the system there was a need to restructure the UI design of the system, the improved UI design proposed in the further sections of the paper show its perks in terms of accessibility of the system.

II. MOTIVATION

There are two main design goals for this system, namely,

1) There is a need to build a highly scalable, abstracted execution environment that supports multi-tenancy. The spike in the use of IoT devices and the growing number of smart cities pushes the developers to steer their designs to efficiently support multiple spikes in the amount of data reaching the servers. A mutli-tenant architecture proves to be the backbone of a successful system in a smart world.
2) To create an easy to use interface to facilitate the interaction of the users with the underlying mesh of edge nodes. A cognitive design of the user interface has statistically proven to increase user efficiency and coming from a Human-Computer interaction point of view, provides intuitive base for a strong sense of directness in the use of a system[1].

III. CHALLENGES

After careful consideration of the current infrastructure and research in the field of IoT devices and building a smarter city, the following areas were identified that needs to be addressed immediately.

1) Eradicate downtime and provide dynamic adaption, migrating live docker containers between edge nodes in a network poses a significant challenge in terms the amount of rerouting of the information from IoT devices to the containers and the inter-container communication.
2) Requirement of a Fault Tolerant architecture: IoT devices are in a constant state of sending data to the cloud servers. This makes the servers a hot spot and cause the server to crash or act less efficaciously. In such a case the users can face significant downtime and loss

of data. A number of architectures have been developed for handling such a scenario like the Raft Consensus Algorithm, the well-known Paxos Algorithm, etc.

3) Dynamic Environment- There is always an assumption made that the environment in which the end nodes are deployed will be stable and will withstand all kinds of adversities. In reality, there are numerous factors that affect the functioning of the edge nodes, they may be a victim of a cyber-attack or any change in the network topology in the deployed area. A resilient and scalable infrastructure is required to withstand unforeseen adversities.
4) Unbalanced cluster load- The overloading of the cluster nodes can cause the re-balance process to make specific application pattern implementation which narrows the choice of workloads that can be hosted in the cluster. This challenge can be tackled using docker container migration.
5) Un-cluttered User interface- The amount of data that is being processed at the edge nodes is huge and presenting this data on the dashboard in an organized way without distracting the user with cluttered, confusing data poses a unique challenge in this project. Displaying data using interactive charts with such huge data is another challenge in this edge cloud solution.

IV. SYSTEM ARCHITECTURE

The extension of Openstack for cloudlet (Kiryong Ha, Mahadev Satyanarayanan), Openstack++ provides multi-tenancy through VM based computing resources. This solution of providing hypervisor based computing resource to establish multi-tenancy adds redundant software abstraction layer to give a non-deterministic application. This project, aims to bring a solution to provide an architecture of a robust platform for the edge cloud that offers multi-tenant, isolated user applications. The Platform provides computing resource through a lighter, more deterministic container based virtualization (IaaS). The computing resource holds all the dependent libraries, runtime environment required to run the isolated application (PaaS). The edge cloud provides a solution offering remote dynamic discovery that enables user to control and manage applications remotely through a web application dashboard (SaaS). A light container based migration scheme is proposed to achieve high availability and near zero downtimes.

A. Server/Cloud

Amazon Web Service (AWS), is used to host a virtual instance of an Ubuntu machine, which is the server that acts as a message router, a database and hosts the dashboard. Refer Fig.1.

B. Edge Node

Client side contains an edge node. An edge node can be any embedded device such as Beaglebone Black, Raspberry Pi, Jetson TX1 and so on. Docker engine is installed on the edge node which allows the users to deploy their application by

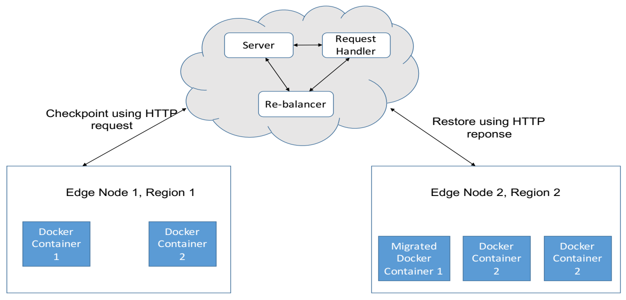


Fig. 1. System overview and Container migration overview

creating containers. Edge node communicates with a WAMP agent to receive the commands issued by users. The response from the edge node is sent back to notify the user about the action taken by the Docker.

C. Dashboard(Work in progress)

The user interface of an application will often make or break it. Although the functionality that an application provides to users is important, the way in which it provides that functionality is just as important. An application that is difficult to use wont be used. It wont matter how technically superior your software is or what functionality it provides, if your users dont like it they simply wont use it. A cognitive design approach is used for building a rich user interface. Dashboard-ing of the project plays an important part while considering the impact of the system on the end user. A box layout is being used for the dashboard development, with focus on streaming live application and container data on the main dashboard screen, providing real-time access to the application and container data. Constantine and Lockwood have described a collection of principles for improving the quality of user interface design. The user interface design for this project tries to satisfy all these principles and provides an easy web interface for accessing the edge cloud management services.

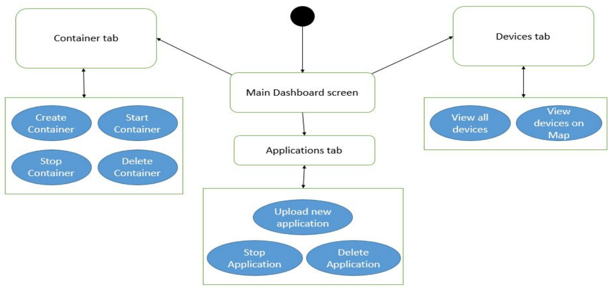


Fig. 2. Dashboard overview

1) UI Design principles:

1) The structure principle- The design of the dashboard has been arranged, on a clear, consistent model that is apparent and recognizable to users, putting related things together and separating unrelated things, differentiating dissimilar things and making similar things resemble one another. Tabbing is done in the dashboard to achieve this

principle, the tasks related to the applications, containers and devices are separated into different tabs, which takes care of just one single type of task.

2) The simplicity principle- The design is simple and articulate in a visual aspect, providing good shortcuts that are meaningfully related to longer procedures. The left hand column providing access to all the major tasks on the dashboard is a feature fulfilling this principle.
3) The visibility principle- The design ensures keeping all needed options for a given task visible without distracting the user with extraneous or redundant information. Tasks like container or application management are simplified by making use of collapsible internal tabs which ensure that only one operation is active and visible to the user at any given time. If the user has to create a new container then he has to click on the create container tab which will open up while other tabs like start, stop, delete stay collapsed and dont confuse the user with overwhelming information.
4) The feedback principle- The design keeps users informed of actions or interpretations, changes of state or condition, and errors or exceptions that are relevant and of interest to the user through clear, concise notifications on the dashboard. The map feature included in the create container tab fulfills this principle, when a user enters a location for the container, the map gets updated with the present location giving user instantaneous feedback on his actions.
5) The reuse principle- The use of the same color scheme makes user co-relate different features of the dashboard, which in turn are reused for similar tasks panned across different tabs. This scheme of reusing internal components and maintaining consistency makes the user remember less things on the dashboard and act intuitively with the design.

2) Flow of UI components: The flow of the UI components of the system is depicted in this section using the images from the working model of the User Interface handling live application data.

1) The Login module, authenticating the user to enter the system. After authentication, the user will be redirected to the main dashboard screen.

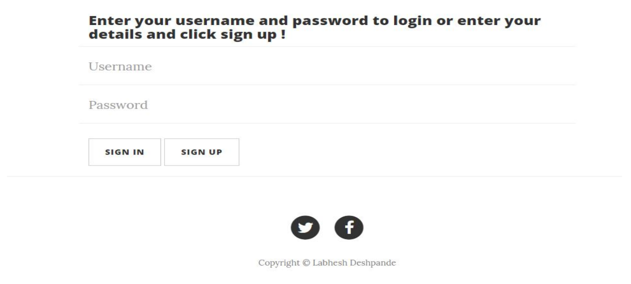

Enter your username and password to login or enter your details and click sign up !

Fig. 3. Login

2) The main dashboard Screen hosting all the tabs for directing all the operations on the system along with Real time CPU utilization vs. Application number chart on the main dashboard screen. Refer Fig.3 and Fig.4.

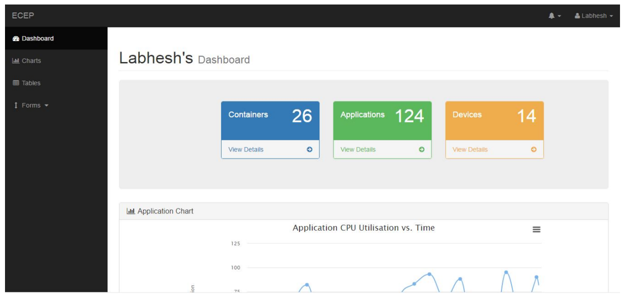


Fig. 4. Main Dashboard Screen

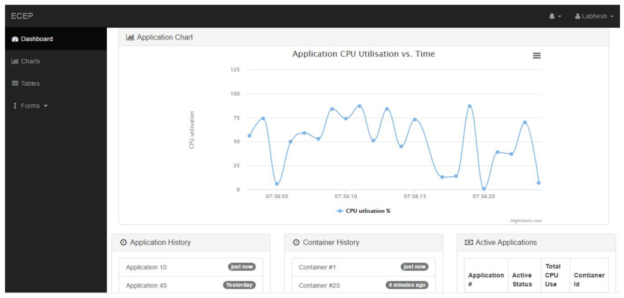


Fig. 5. Live chart of CPU utilization vs.Time

3) The container management tab, dictating all the operations on the containers of the particular user. Refer Fig.5 and Fig.6.

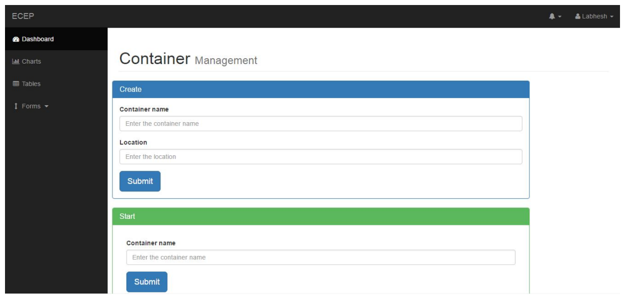


Fig. 6. Container management tab

4) The devices tab, handles all the operations on the devices for the particular user. Refer Fig.7.

D. Container Management

Containers provide a complete software package required to run a users applications on the edge node by abstracting the underlying operating system. Containers are launched with applications requested by each user. All dependent libraries to run the application inside a container will be installed before executing an application. Every user can deploy new container for each application and later obtain the results. Once the

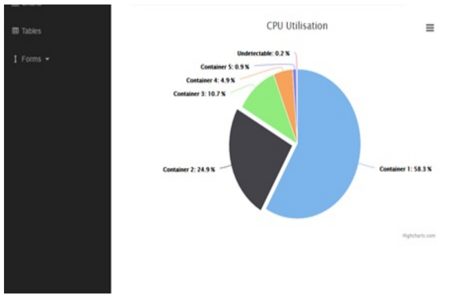


Fig. 7. Container management tab

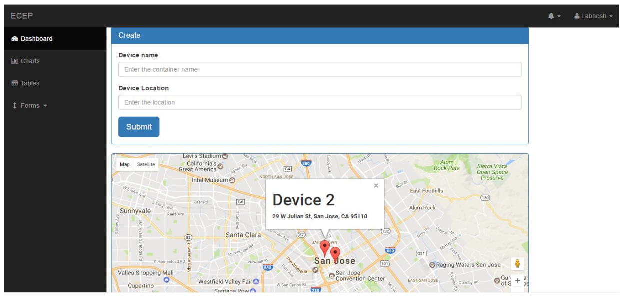


Fig. 8. Device management tab

execution of an application is completed, then containers are deleted automatically by the Docker daemon.

V. Container Migration(Work in progress)

Live container migration refers to the process of migrating containers between different hosts or clouds without disconnecting the client. The memory file system, network connections running on a hardware is transferred from one machine to another preserving the state without downtimes. The pre-copy container migration scheme is being proposed to be used for container migration. In this the platform turns track of the source node and copies this memory in parallel on destination node. After that, it freezes the container, gets the rest of the state, migrates it to a destination node, restores and unfreezes it. Container migration module proposes to increase the availability of data by migrating live containers between different hosts when one of the host fails or is overburdened with requests. Load balancing on unbalanced cluster nodes is also handled using container migration. We propose a two-step process to achieve container migration,

1) Using MQTT Protocol for Container Orchestration: MQTT or Messaging Queue Telemetry Transport is a light weighted and simple protocol. It is considered ideal for IoT as it takes only 80 bytes to connect to a server and stay connected. Publish and consumption of messages is about 20 Bytes. Once a connection is built, there isn't a need to tear it down hence reducing overhead drastically. The reason for not the simple HTTP for this purpose is that HTTP is stateless which makes it a great choice for request response type architecture but its text based source information doesnt allow it to push data. Pertaining to our use case to orchestrate containers on edge nodes. We propose the installation of MQTT broker on the Raspberry Pi. This is considered the master. The edge node connects to this broker which in turn connects to the Global Master (On the cloud). Other Raspberry pi hosts in the network will be termed agents and will be required to migrate the container. The global master keeps IP addresses, images, container IDs, and topics of all the hosts in the network. The master orchestrates the activity of container migration by publishing the IP address of the Free Node over the MQTT broker.
2) Using CRIU integration with Docker : A top level checkpoint sub-command in Docker should be used to create a new checkpoint, and list or delete an existing checkpoint. These checkpoints are stored and managed by Docker, unless you specify a custom storage path. The master will now checkpoint a live container using the checkpoint command in docker. This will halt the current process and no new process logs will be printed. To restore the container the master will redirect messages through MQTT to the free node in the network using the restore command. The restore command can be used to restore the container into a completely different container and thus achieving our goal of container migration.

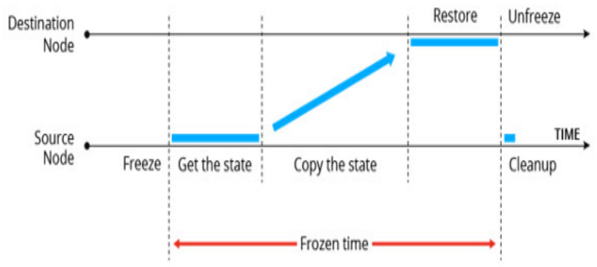


Fig. 9. Container Live Migration[4]

VI. Evaluation

The performance evaluation is performed by deploying a server on AWS and considering Beaglebone Black as the end node. The following performance is impacted by the network upload and download bandwidth. In our analysis the upload bandwidth is observed to be 11.45 Mbps and download bandwidth is observed to be 24.2 Mbps.

A. *Performance for creating a remote container by downloading a new image from the cloud*

. Fig. 10 shows the performance for creating a remote container by downloading a new image from the cloud. This is the first step when the user want to deploy their application to the remote edge node. The user can select the type of the container image and push it the edge node. The Fig. 10 represents the time required to pull an image from Docker hub in the cloud and create a container in the edge node from the downloaded image. For performance analysis purpose, we considered different 5 images with size ranging from 1.5kB to 350MB. The calculated time includes the downloading time of the image and the time of creating a container out of it. Network latency is assumed to be equal during the course of evaluation.

To pull hello-world image, which is of 1.8kB in size from Docker hub and create a container it took approximately 3.75 secs. Upon issuing the command, Docker daemon initials looks for the image in the cache and if doesn't find it then will pull the latest image from the hub and creates a container using this image. Similarly, the "training/postgres" image consumes a time of just over a minute to pull and deploy a container. When compared with the small size image, there is an exponential increase in the time factor. The reason behind this is that the Docker daemon requires more time to create an image for large packages in additional to the linear downloading time. Fig. 10 clearly suggests that, bigger image sizes require more overhead time including downloading and creating a container of that image.

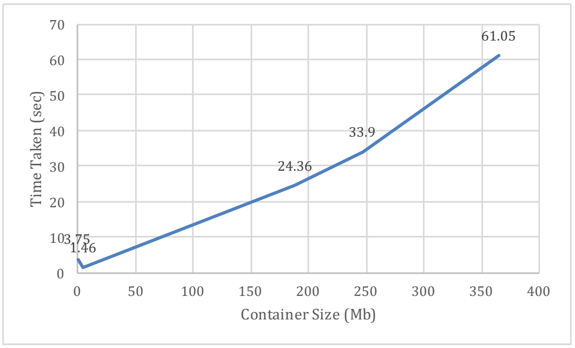


Fig. 10. The time taken to create container for new images with different size.

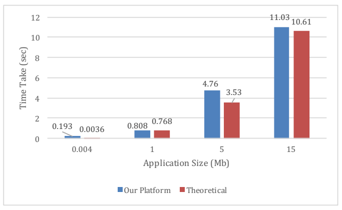
B. Upload Applications to the Edge Node

To evaluate the performance of uploading an application file from user dashboard to webserver, we perform time consumption calculation based on a fixed network bandwidth (11.45Mbps as the upload speed). The network bandwidth is assumed to be constant over the course of evaluation.

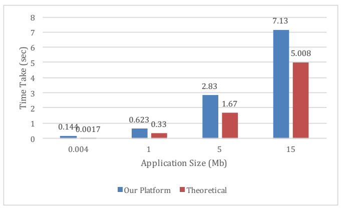
The web server reads the compressed application file using "multer" and uploads serially through a HTTP request. The uploaded file is then moved to a unique storage space with unique ID. When the file is too big, we will divide them into several small compressed files, then these compressed files will be transferred. The total time consumed includes the file compression, division, and transfer. As shown in Fig. 11a, the results show that the time consumed was almost linear with increase in application files. The difference in time observed is due to the time taken by our platform to process the data, create a directory, upload the file and update the database.

C. Getting the log file from the edge node

As user requests to fetch the log file for the debugging purpose, a command from the cloud dashboard will be transferred



(a) To the web server



(b) To the edge node

Fig. 11. The time taken to transfer the application file: (a) to the web server; (b) to the edge node.

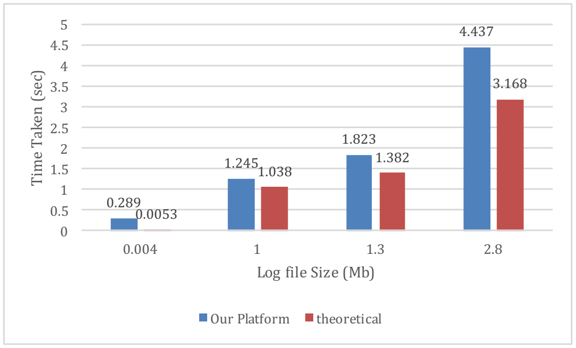


Fig. 12. The time taken to transfer the log file from the edge node to the cloud dashboard.

to the end node, and fetches the log result file from end node to the cloud dashboard. The transfer of file from end node to webserver is achieved with the help of HTTP protocol. The log file available on the end node is read in the form of chunk and is transmitted from end node to cloud dashboard. The received chunk is copied onto a file on the server until the end of file. Different sized result files were considered to measure time and the time taken. Upload speed of 11.45Mbps is considered during the evaluation. Fig. 12 shows the required time to fetch the log file with respect to different file sizes. The difference in time is due to the time taken by our platform to send a message from webserver to end node, process the request, copy the file from the container to the local path, locate the file and then transfer to the cloud. This analysis measures the time required for the complete procedure which constitutes of, send a command from user dashboard, route the command to a particular edge node, copy the file from container to the local path on the edge node, transfer the file from to the server, point to the messaging service lay to download and transfer of file to the user dashboard.

VII. Related Work

There have been numerous publications relating to the issue being handled through this project, majority falling in the following category:

1) Cloud Computing [1], [2], [3], [4]: Vision, Architecture and Characteristics. The rapid growth of using cloud-based services in recent years is an impossible fact to be denied as it has increased the efficiency in accessing to

shared pools of configurable computing resources. According to this rapid growth, it is anticipated that cloud computing will be the most important and challenging issue in IT industry.

2) Edge Computing [5], [6], [7]: Vision and Challenges. Cloud computing can get strained to its limits with peta bytes of data being produced in the coming time and hence the need of a conglomeration of edge computing with it is an emerging requirement to provide low latency for systems without compromising the efficiency of the system.
3) A survey of fog computing [8]: concepts, applications and issues. Despite the increasing usage of cloud computing, there are still issues unsolved due to the inherent problem of cloud computing such as unreliable latency, lack of mobility support and location-awareness. Fog computing, also termed edge computing, can address those problems by providing elastic resources and services to end users at the edge of network, while cloud computing are more about providing resources distributed in the core network.

VIII. Conclusion

We aim at leveraging the minimal usage of network bandwidth by asynchronous communication between server and client. Docker technology provides a lightweight virtual space in the form of containers. Containers consume less memory when compared to a virtual machine, which is an advantage due to the memory constraints in embedded environment. These containers can communicate over the network which allows us to remotely orchestrate it through a dashboard. The use of an effective dashboard improves the utility of the software application and the following the UI design principles mentioned in the paper, improves the accessibility of the user interface, making it more user friendly. The ongoing work on container migration will enhance the present architecture of the system and strive towards achieving fault tolerance with almost zero downtimes.

Acknowledgment

I would like to thank Dr. KaiKai Liu for his constant guidance in this project. Dr. Liu's valuable inputs on the topic of edge computing and container migration encouraged me to research on these topics and think about prospective ways to include container migration in the project,Edge Computing Embedded Platform.

References

[1] A. V. Dastjerdi, H. Gupta, R. N. Calheiros, S. K. Ghosh, and R. Buyya, "Fog computing: Principles, architectures, and applications," *arXiv preprint arXiv:1601.02752*, 2016.

[2] 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.

[3] R. Mora, "Cisco iox: Making fog real for iot, blogs@ cisco-cisco blogs, june 2015."

[4] Z. Pang, L. Sun, Z. Wang, E. Tian, and S. Yang, "A survey of cloudlet based mobile computing," in *Cloud Computing and Big Data (CCBD), 2015 International Conference on*. IEEE, 2015, pp. 268–275.

[5] O. Sefraoui, M. Aissaoui, and M. Eleuldj, "Openstack: toward an open-source solution for cloud computing," *International Journal of Computer Applications*, vol. 55, no. 3, 2012.

[6] A. Solano, R. Dormido, N. Duro, and J. M. Sánchez, "A self-provisioning mechanism in openstack for iot devices," *Sensors*, vol. 16, no. 8, p. 1306, 2016.

[7] K. Ha and M. Satyanarayanan, "Openstack++ for cloudlet deployment," *School of Computer Science Carnegie Mellon University Pittsburgh*, 2015.

[8] G. I. Klas, "Fog computing and mobile edge cloud gain momentum open fog consortium etsi mec and cloudlets," 2015.