Evaluating Performance of In-Situ Distributed Processing on IoT Devices by Developing a Workspace Context Recognition Service

Jose Paolo Talusan, Francis Tiausas, Sopicha Stirapongsasuti,
Yugo Nakamura, Teruhiro Mizumoto, Keiichi Yasumoto
Nara Institute of Science and Technology, Japan
Email: {talusan.jose_paolo.tg3, tiausas.francis_jerome.ta5, stirapongsasuti.sopicha,
nakamura.yugo.ns0, teruhiro-m, yasumoto}@is.naist.jp

***Abstract*—With the number of IoT devices expected to exceed 50 billion in 2023, edge and fog computing paradigms are beginning to attract attention as a way to process the massive amounts of raw data being generated. However, these paradigms do not consider the processing capabilities of the existing commodity IoT devices in the wild. In order to solve this challenge, we are developing a new middleware platform called IFoT, which processes various sensor data while considering Quality of Service (QoS) by utilizing the computational resources of heterogeneous IoT devices within an area. This allows smart services to be created and processed in parallel by various IoT devices. In this paper, we show the effectiveness of the IFoT based approach of constructing services. We designed and implemented a workspace context recognition service, utilizing environmental sensor data processed in a distributed manner according to the IFoT framework. We evaluate the QoS of IFoT middleware and its feasibility when used on commodity devices such as the Raspberry Pi, through the service.**

***Index Terms*—Edge Computing, Internet of Things, Middleware, Distributed processing, Activity recognition.**

I. Introduction

Internet of things (IoT) devices continue to be created and deployed at an increasing rate. The number of IoT devices in the real world is expected to reach 50 billion by 2023 [1]. With the increase in IoT devices, come the increase in raw data, most of which is consumed and processed by companies such as Google, Apple and Facebook, then provided to users as services that increase quality of life and social media. One of the key issues in a world that is saturated by the ever increasing data is how to gather, process and aggregate it with low latency and low cost.

Cloud computing is currently the main platform used for deploying IoT services [2]. However, not all cloud-based approaches may be suitable for services targeted for smaller communities without suitable access to the global IoT, or for private and secure services whose data cannot be given to global IoT systems.

Now edge and fog computing paradigms are attracting attention with their ability to process data much closer to the source. In edge-based existing studies [3], [4], edge clouds are deployed and used in a metropolitan based environment to process tasks with low delay. However utilization of computational resources and use of resource constrained devices are not taken into consideration.

Therefore a new data processing framework called the Information Flow of Things (IFoT) [5], [6] is proposed for processing information flow from various IoT devices in a timely and scalable manner based on distributed processing on in-situ devices. The IFoT framework flexibly utilizes computation resources of IoT devices existing near the data source, aiming to efficiently coordinate and maintain smart community services with low cost and low latency.

In order to realize the IFoT framework, we are developing a new middleware platform (IFoT middleware) [6]. In the IFoT middleware, heterogeneous IoT devices are grouped into clusters and configured to work together to satisfy the computational demand of services. Services are user created applications that process and convert raw data into usable ones. User queries and requests of services, processed through the platform, need to meet a predetermined service level agreement requirements (SLA). To be able to satisfy a service's SLA, the IFoT middleware leverages the cluster's distributed processing capabilities.

To demonstrate the effectiveness of the IFoT middleware, in this paper, we create a use case and deploy a service that utilizes the middleware. We design a workspace context recognition service that can be used by offices or buildings with many rooms. This service use environmental sensors and commodity IoT devices deployed in various rooms. The target scenario is suitable on IFoT middleware because it is able to demonstrate how many heterogeneous IoT devices can be used to provide services with minimal latency due to its distributed configuration.

We developed and implemented a distributed machine learning mechanism to utilize the cluster's computational resource [7]. This mechanism is modified and improved in this paper using a scheduling system which utilizes the in-memory data structure. With this improvement we were able to generate responses for multiple queries in 2.3 seconds which can then be decreased linearly the more nodes are added.

In the following Sections II and III, we discuss prior work

978-1-5386-9151-9/19/$31.00 ©2019 IEEE

and the IFoT platform, respectively. In Section IV, we describe in more detail how we implement and deploy distributed processing of tasks for a workspace context recognition service within the platform. In Section V, we show the actual setup and configuration and the results of the experiments. Finally, we conclude the paper in Section VI.

II. Related Work

A. Edge and Fog computing

Fog Computing [8] and Edge Computing [9] are paradigms that mitigate server load by processing data on servers nearer to the data source. Edge computing may act as a bridge between IoT devices and the cloud. Edge and fog computing make it possible to minimize the latency of tasks compared with the cloud. These platforms perform roles such as IoT device management, network management and data processing and transferring. While edge computing is able to minimize latency as well as efficiently use available bandwidth, it still faces challenges with regards to data partitioning and offloading of tasks.

B. Docker Technology

Containerization [10] is technology that is now popularized by Docker. Docker is a light-weight application that facilitates the creation of self-contained environments called containers [11]. These containers are isolated instances similar to the solution provided by virtual machines (VM). However, unlike VMs they achieve minimal overhead by sharing the kernel with the host OS. Docker containers enable one to build dynamic services which can then be distributed within a cluster of devices through an orchestrator. Docker has been used in research that aims to deploy services on single-board computers [12] [13], as well as in research on cost-efficient edge frameworks [14].

C. Middleware platforms

While distributed computing through IoT devices using containers have been an attractive prospect, there is still the challenge of heterogeneous IoT devices. Research is being done to address where current frameworks fall short in dealing with the heterogeneity of distributed computing. Schafer et al. in Tasklets [15] and Bhave et al. [16] attempted to ease the burden of heterogeneity for distributed and edge computing by using middleware and virtualization technologies to efficiently handle multiple heterogeneous devices and tried to pool their computation resources together.

While studies on containerization and middleware platforms have been found feasible and successfully done on commodity devices such as Raspberry Pi [13], [17], [18], and have been able to provide some form of distributed processing, these systems do not focus on providing service for users within an area. Also, these prior systems and platforms while being able to create a middleware on heterogeneous devices, were not utilizing the computational resource of the pooled devices for more sophisticated data processing/analysis by means of distributed machine learning.

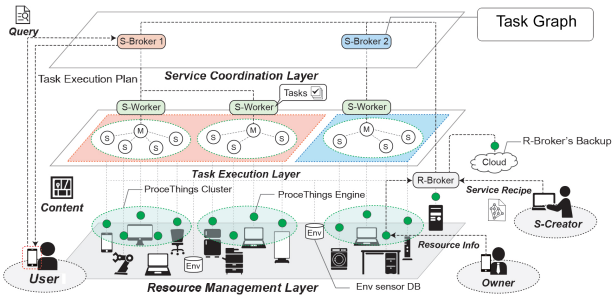

Fig. 1. Overview of IFoT Middleware Platform Architecture

III. IFoT Middleware Platform

The IFoT middleware platform (parts of its mechanisms are presented in [6]) is a concrete realization of IFoT framework. It consists of three main layers: *Resource Management Layer, Task Execution Layer and Service Coordination Layer*. All IoT devices in the platform are considered nodes and they may serve different functions within the architecture shown in Fig. 1.

A. Platform Architecture

1) Resource Management Layer*:* Manages IoT devices that participate in the platform. It consists of the *resource broker (R-Broker)*. It is manually set by community as (typically) the most powerful node in the network and has information on all available nodes. It also manages all the nodes in the platform.

The R-Broker handles resource registration by resource owners through a web interface. The owners provide information of IoT devices such as processing capabilities, available sensors and location details to the resource broker. The registered nodes are then configured into Docker nodes, to be configured into either *Service Brokers* or *Service Workers* depending on their computational capability, comprising the service coordination layer and the task execution layer explained below.

2) Service Coordination Layer*:* Handles the communication between end-user and the task execution layer. It consists of the *service brokers (S-Brokers)*. The S-Brokers manage services. Each S-Broker is the gateway by which users query the service through a web interface. S-Broker is assigned manually by a service creator to (typically) the most powerful node available after the R-Broker. S-Broker manages multiple S-Workers within its service area and adds more S-Workers/worker nodes to its manageable resource pool to provide needed QoS level for the current computation demand (i.e., the number of queries per unit of time) [6].

3) Task Execution Layer*:* Handles the execution of services or task graphs that the platform offers the users. It consists of *service workers (S-Workers)*. This layer is configured to function as a cluster and is designed to execute tasks in a distributed manner.

Once nodes are registered into the platform, they are setup as clusters for a specific location. *Clusters* are the task

execution block of the platform which are composed of the following:

a) Service Worker (S-Worker): S-Worker is the virtual representation of the task execution cluster. It is formed by a Docker Swarm cluster which consists of a master node and multiple worker nodes. They are in charge with communicating with the S-Brokers regarding the user queries for task execution. Upon receiving requests, they are the ones that manage the task distribution to multiple worker nodes.

b) Master Nodes: handle the task distribution to the multiple worker nodes. Databases such as the environmental sensor databases, are placed on master nodes as well. In our implementation these are deployed on Raspberry Pi.

c) Worker Nodes: are the basic execution nodes of the platform. They are tasked with executing tasks allocated by the master node. Each worker node is a single IoT device with limited or constrained computational resources. In our implementation worker nodes are Raspberry Pi.

d) Environmental sensor database (envDB): are time series database that collects and aggregates data from the sensors connected to the platform. Services such as activity recognition will gather data from envDBs. These are assigned to the master nodes of S-Workers.

IV. Workspace Context Recognition Service

A. Service Scenario

In this section, we present a workspace context recognition service scenario as a typical use case of IFoT middleware platform.

We assume that future smart offices will have many *free address workspaces* where many environmental sensors are installed as well as having their own network infrastructure. Employees can freely use these rooms for meetings, work, and recreation at anytime. However, if the office space is too large, it is difficult for employees to figure out which workspace is currently available and suitable for their needs. This is our motivation to develop the workspace context recognition service.

Workspace Context Recognition Service: Using this service, office employees can get useful information regarding the room context such as the comfort level, noisiness, and the possible over use or under use of certain rooms. This information is generated through the data processing (Statistical processing and machine learning inference using pre-trained models within the platform) of data from environmental sensors which are located in the rooms. This service can be accessed by the employees through the S-Broker on the local intranet. All information remains private since data is only stored locally within the nodes.

The number of rooms that need to be monitored as well as people monitoring these rooms, affect the performance of the platform. Performance can be increased by adding more IoT nodes into the system, increasing the local computation resource, thus avoiding the need for a more powerful central terminal. Clusters of commodity single-board computers such as Raspberry Pis are more than enough for this application.

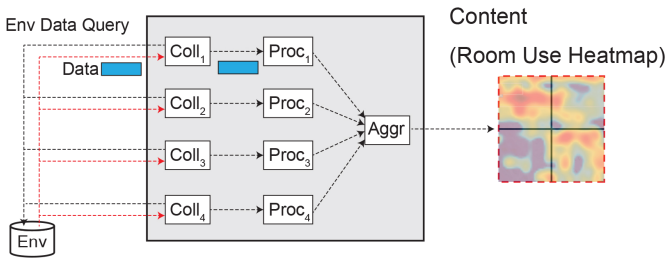

Fig. 2. Task graph for workspace context recognition service

Furthermore, having more nodes within the platform allows the creation of more services that may be needed by employees in the office space. Addition of a service is simply a case of adding the required sensors and a task graph that details the collecting, processing and aggregating tasks.

Other use cases: This framework can be generalized to similar use cases which feature characteristics such as: geo-spatial sensor data, machine-learning, heatmap, etc. Modifications to the task graph allow the framework to handle such use cases, without the need for large changes of the entire framework.

B. Details of the Task graph

Task graphs (or service recipes) detail how a service handles queries by users. Each service has a corresponding task graph. The task graph for the workspace context recognition service is shown in Fig. 2. Sensor data is stored inside an envDB located inside the monitored room. Upon receiving a query, the S-Broker, executes a task graph that is executed by the S-Worker(s). The task graph makes use of Redis [19], an open source in-memory data structure that functions as the main queueing system of the IFoT middleware. Tasks are queued onto Redis and then worker nodes monitor these queues and process them when they are free.

1) Collecting Task: For the current experiment, the query includes time information for the test room. This time information is then sent to a worker node which in turn collects the data from the envDB inside the room. This task is executed in parallel for each received query. The worker obtains the data in the form of a JSON string which is passed onto the queue for processing.

2) Processing Task: Processing tasks will be done in parallel, depending on the number of available worker nodes that monitor the queue. Upon receiving the JSON string from the queue, it converts these to a Pandas dataframe. It also loads the pre-trained classification model that is stored in each worker node for use. It will then use the loaded model to classify the status of the room based on the sensor data. The worker node then sends the classified labels back to the queue for the aggregator.

3) Aggregation Task: This task is usually done on one worker node. It will wait either for all nodes to finish or set a timeout. Upon gathering all the coordinate and label information, it will generate a heatmap that specifies the usage of a particular area. This is then saved as an SVG image and then sent back to the S-Broker for displaying back to the user.

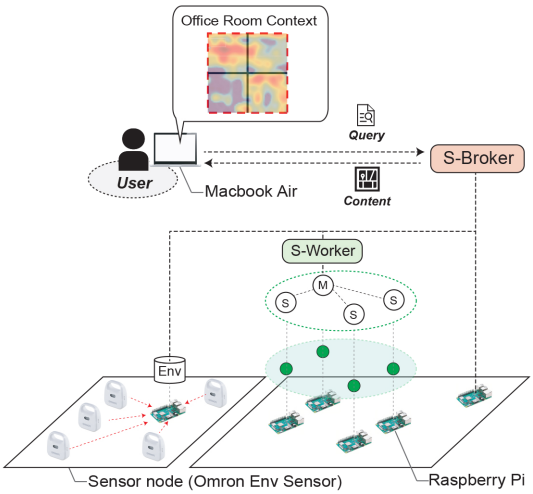

Fig. 3. Current experiment architecture

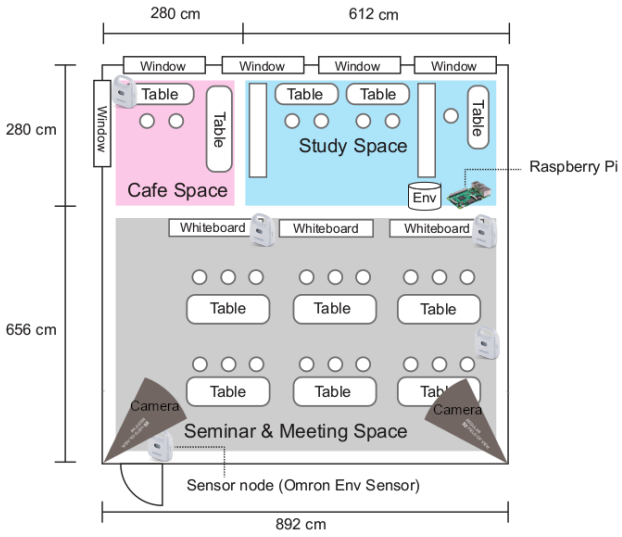

Fig. 4. Placement of Environmental sensors in the implementation

V. Implementation and Evaluation

A. Implementation

The smart room context recognition service with the IFoT middleware platform is built using Raspberry Pi 3 Model B (Pi) and Omron 2JCIE-BL01 Environmental Sensor. The Pi is equipped with 1.4GHz ARMv8 processor, 1GB DDR2 SDRAM, Wifi and Bluetooth low energy (BLE) connectivity. It uses Raspbian operating system, a version of Linux Debian, optimized for ARM. The sensor is a wireless sensor that is equipped with 7 different sensors: temperature, humidity, light, UVI, absolute pressure, noise and acceleration. The Omron sensor measures data at 300 second intervals (this period can be set between 1 second and 1 hour), at this rate lifetime of the battery is 3 months. Each period, the sensor obtains the room's current temperature, relative humidity, ambient light, UV index, pressure and sound noise. All these information are sent to the Pi node, which listens to the sensor beacons via Bluetooth Low Energy. It receives the sensor information as well as timestamp, RSSI, sensor MAC address, gateway address, estimated distance via the RSSI, heat stroke factor, discomfort index and battery level, which is stored as a row with 18 columns in the envDB.

Five environmental sensors were placed in a large multi-function room used for seminars, meetings, recreational activities, and discussions. Due to the room's size, it was further broken down into several areas as shown in Fig. 4. The locations of these sensors were chosen to maximize the amount of data being collected on the varying use of the room throughout the day. Data is broadcast by the sensors every 5 minutes and were received by a sensor node (Raspberry Pi) located in the same room. This sensor node is equipped with an envDB for storing the timeseries data generated by all the sensors. Transmission of data is through Bluetooth Low Energy and thus RSSI information was also recorded. Two Raspberry Pi Camera Module v2 cameras were setup in two corners of the room, as shown in Fig. 4, to capture ground truth data during the experiment. We then manually label each 10 minute interval based on the number of people present. We use 4 classification levels No Use, Low Use, Medium Use, High Use. We set these based on the number of people present in the room. 0: [No Use], 1-3: [Low Use], 4-9: [Medium Use] and 10+: [High Use]. This was in an effort to keep training and classification simple since the experiment location was a single multi-functional room. Subsequent experiments with multiple rooms may increase the classification to take into account more classes.

The classification model is trained using a Support Vector Classification (SVC) algorithm via Scikit-Learn package. The features that are used from each sensor are humidity, light, noise, RSSI, and temperature. The other three features, acceleration, UVI and pressure showed little to no changes for the duration of the experiment and were not used.

Since all 5 sensors were placed in a single room at different locations, we used each sensor's 5 different sensor data as columns. Raw sensor readings were used without further modification for the data set. This results in a 26 column dataset (including the timestamp which is used as a feature) with almost 2000 rows for 4 days of data gathering. This was trained offline using the SVC algorithm. The resulting machine learning model which has an accuracy of 62% as shown in Fig. 5, is then saved into a file for distribution to the S-Workers.

In our implementation, the model is sent first to the S-Broker and is then automatically distributed to the S-Worker via Python script. S-Worker, located outside the room simply for debugging purposes, is connected to the university's network via wired connection. The sensor node, which contains the envDB, is connected to the same network via WiFi. The sensor node can also be a worker node of the S-Worker, however for simplicity, it was configured separate from the S-Worker. In future experiments, this will be made part of the S-Worker. S-Workers and S-Brokers connect to each other through the same network above, via wired connec-

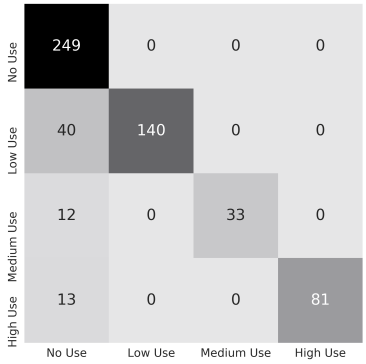

Fig. 5. Confusion matrix for SVC model

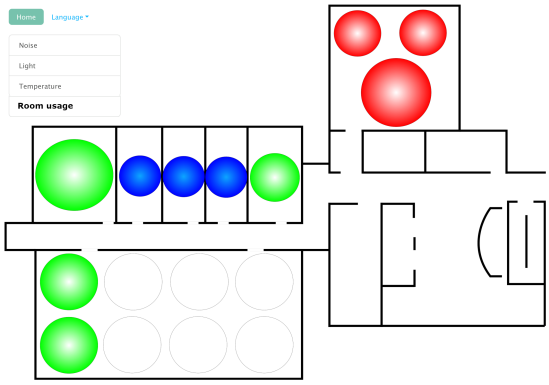

Fig. 6. Demonstration of possible output of the Smart Room context recognition service, where blank(no use), blue(low use), green(medium use) and red(high use)

tion. The user can access the S-Broker through any method (wired/wireless, smart phone or PC) as long as they are connected to the local area network of the university. In this case, each worker node receives a copy of the same model. Upon receiving a user query, the S-Broker, executes a task graph as shown in Fig. 2. The classification process is then executed in the following manner: (1) Upon receiving service requests, the S-Broker divides it into singular room queries. (2) It obtains and forwards the time information of each room query into the queue. (3) The S-Worker monitors this queue and assigns the tasks to a free worker node under it. (4) The worker node performs the collecting task and then using the previously received pre-trained model, the processing task. (5) Upon classification, it then sends the labeled data as well as other room information data back into a queue. (6) Again, the S-Worker, monitoring the queue, assigns this to a free worker node to perform the aggregation task as detailed in Sec. IV-B The next section discusses the evaluation of this task execution and the effects of the S-Worker on the QoS.

B. Evaluation: Centralized vs. Distributed Task Execution

Given the setup above, a use case was imagined for the service: employees want to know information on the workspaces in the building. They query the S-Broker for information based on the sensors deployed in each space. The number of rooms or number of other entities (e.g., other building staff, local fire department monitors, etc.) regularly performing such a query at the same time may vary, leading to scenarios that require a serviceable quality of service (QoS) from the platform. The output of a user query is a corresponding label for the room they are querying, in the future this output can be displayed in the form of a heatmap as shown in Fig. 6, where room usage is shown in various colors.

Since we only have sensors placed in a single room, we simulate how the system would behave when multiple rooms are being queried at once. To be able to do that, we randomly select 100 data points (i.e., 100 samples) from the dataset and set it as the target of query. Since the sensors would be the same regardless of the room which it is placed in, we then suppose that each row (a data point) in the data set is a different room. Based on the study of Egger et al. [20], there is a direct relationship between delay and dissatisfaction, with this we aim that the workspace context recognition service should be able to return a response within around 2 seconds.

We perform the following experiment to investigate the QoS the platform is capable of delivering. We test the system's ability to handle 100 rooms being queried at once. We first configured the platform such that the S-Worker would only run 1 worker node and then increase the number of nodes via the scale-out method detailed in [6].

We consider the total execution time as the time measured from when the user sent the query until the response of the heatmap is received by the same user. Fig. 7 shows the QoS of the platform against the number of worker nodes present in an S-Worker. Given large sets of data bound in a single query, a single node on average, total execution time goes beyond the set limit of 2 seconds and fails to achieve acceptable QoS. Increasing the number of nodes to 5, allows the platform to respond with an average of 2 seconds for large data set queries. This total execution time only goes lower the more nodes are added.

Next we simulate the effect of in-situ resource provisioning with scale-out [6], an additional implementation for the IFoT platform. We overload a single node using the same 100 rows used in the previous experiment, but we divide the 100 rows into individual queries and then send simultaneous queries to S-Broker. As seen in Fig. 8, using a single node, it has an average total execution time of 25 seconds. With an initial 4 nodes, we can decrease this total execution time to 5 seconds, quicker than a single node but still unable to meet the QoS. Finally, implementing in-situ resource provisioning to the nearest 3 neighbor nodes, we can decrease the total execution time to 1.2 seconds.

VI. Conclusion

In this paper we designed and developed a use case for the IFoT middleware platform. The use case of smart room context recognition system is implemented into the platform as a service. Users query the service in order to identify the usage of workspaces in an office environment. The main goal

Fig. 7. Execution times for large sets of data in single queries with varying number of workers

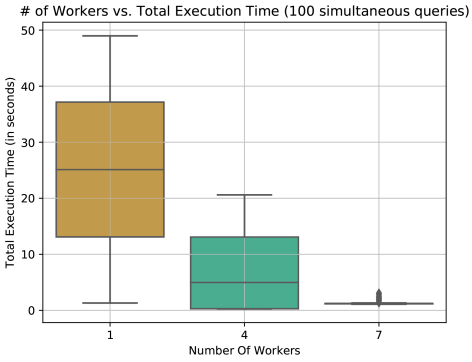

Fig. 8. Execution times for small sets of data in multiple parallel queries with varying number of workers

for the service is to be able to provide a certain QoS such that the service is able to respond to multiple queries within 2 seconds. We achieve this by using the IFoT middleware platform that uses pre-trained machine learning algorithms and computational resources right at the data source to classifiy and recognize room usage using environmental sensors. In addition, we implement distributed processing on the platform, this improves the QoS of the system from a total execution time of 4.2 seconds to less than 2 seconds. Furthermore, with the implementation of adaptive in-situ resource allocation, we show that we can further improve the total execution time for 100 simultaneous requests from 4 seconds to 1.2 seconds.

The platform was implemented on a single room in the essence of saving time, but we show that this system is feasible and is able to deliver acceptable QoS regardless of the number of rooms being monitored.

Acknowledgement

This work was in part supported by JSPS KAKENHI Grant Number 16H01721 & 17J10021 & 26220001 and R&D for Trustworthy Networking for Smart and Connected Communities, Commissioned Research of National Institute of Information and Communications Technology (NICT).

References

[1] Iot: number of connected devices worldwide 2012-2025 — statista. [Online]. Available: https://www.statista.com/statistics/471264/iot-number-of-connected-devices-worldwide/

[2] J. Gubbi, R. Buyya, S. Marusic, and M. Palaniswami, "Internet of things (iot): A vision, architectural elements, and future directions," *Future generation computer systems*, vol. 29, no. 7, pp. 1645–1660, 2013.

[3] L. Tong, Y. Li, and W. Gao, "A hierarchical edge cloud architecture for mobile computing," in *INFOCOM 2016-The 35th Annual IEEE International Conference on Computer Communications, IEEE*. IEEE, 2016, pp. 1–9.

[4] L. Wang, L. Jiao, J. Li, J. Gedeon, and M. Mühlhäuser, "Moera: Mobility-agnostic online resource allocation for edge computing," *IEEE Transactions on Mobile Computing*, 2018.

[5] K. Yasumoto, H. Yamaguchi, and H. Shigeno, "Survey of Real-time Processing Technologies of IoT Data Streams," *Journal of Information Processing*, vol. 24, no. 2, pp. 195–202, 2016.

[6] Y. Nakamura, T. Mizumoto, H. Suwa, Y. Arakawa, H. Yamaguchi, and K. Yasumoto, "In-situ resource provisioning with adaptive scale-out for regional iot services," in *Proceedings of the Third ACM/IEEE Symposium on Edge Computing (SEC 2018)*, 2018, pp. 203–213.

[7] J. P. Talusan, Y. Nakamura, T. Mizumoto, and K. Yasumoto, "Near cloud: Low-cost low-power cloud implementation for rural area connectivity and data processing," in *2018 IEEE 42nd Annual Computer Software and Applications Conference (COMPSAC)*, vol. 02, July 2018, pp. 622–627.

[8] 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*, ser. MCC '12. New York, NY, USA: ACM, 2012, pp. 13–16.

[9] P. Garcia Lopez, A. Montresor, D. Epema, A. Datta, T. Higashino, A. Iamnitchi, M. Barcellos, P. Felber, and E. Riviere, "Edge-centric computing: Vision and challenges," *ACM SIGCOMM Computer Communication Review*, vol. 45, no. 5, pp. 37–42, 2015.

[10] . Kovcs, "Comparison of different linux containers," in *2017 40th International Conference on Telecommunications and Signal Processing (TSP)*, July 2017, pp. 47–51.

[11] C. Boettiger, "An introduction to docker for reproducible research, with examples from the R environment," *CoRR*, vol. abs/1410.0846, 2014. [Online]. Available: http://arxiv.org/abs/1410.0846

[12] A. van Kempen, T. Crivat, B. Trubert, D. Roy, and G. Pierre, "Mec-conpaas: An experimental single-board based mobile edge cloud," in *2017 5th IEEE International Conference on Mobile Cloud Computing, Services, and Engineering (MobileCloud)*, April 2017, pp. 17–24.

[13] C. Pahl, S. Helmer, L. Miori, J. Sanin, and B. Lee, "A container-based edge cloud paas architecture based on raspberry pi clusters," in *2016 IEEE 4th International Conference on Future Internet of Things and Cloud Workshops (FiCloudW)*, Aug 2016, pp. 117–124.

[14] M. Al-Rakhami, M. Alsahli, M. M. Hassan, A. Alamri, A. Guerrieri, and G. Fortino, "Cost efficient edge intelligence framework using docker containers," in *2018 IEEE 16th Intl Conf on Dependable, Autonomic and Secure Computing (DASC)*, Aug 2018, pp. 800–807.

[15] D. Schafer, J. Edinger, J. M. Paluska, S. VanSyckel, and C. Becker, "Tasklets:" better than best-effort" computing," in *Computer Communication and Networks (ICCCN), 2016 25th International Conference on*. IEEE, 2016, pp. 1–11.

[16] S. Bhave, M. Tolentino, H. Zhu, and J. Sheng, "Embedded middleware for distributed raspberry pi device to enable big data applications," in *2017 IEEE International Conference on Computational Science and Engineering (CSE)*, vol. 2, July 2017, pp. 103–108.

[17] Y. Elkhatib, B. Porter, H. B. Ribeiro, M. F. Zhani, J. Qadir, and E. Rivire, "On using micro-clouds to deliver the fog," *IEEE Internet Computing*, vol. 21, no. 2, pp. 8–15, Mar 2017.

[18] X. Wang, S. Jiang, X. Xu, Z. Wu, and Y. Tao, "A raspberry pi and lxc based distributed computing testbed," in *2016 6th International Conference on Digital Home (ICDH)*, Dec 2016, pp. 170–174.

[19] Redis. [Online]. Available: https://redis.io/

[20] S. Egger, T. Hossfeld, R. Schatz, and M. Fiedler, "Waiting times in quality of experience for web based services," in *2012 Fourth International Workshop on Quality of Multimedia Experience*, July 2012, pp. 86–96.