Here, There, Anywhere: Profiling-Driven Services to Tame the Heterogeneity of Edge Applications

Manish Pandey[§]
School of Computer Science and Engineering
Kyungpook National University
Daegu, South Korea
manish@knu.ac.kr

Breno Dantas Cruz[§]
Software Innovations Lab
Virginia Tech
Virginia, USA
bdantasc@cs.vt.edu

Minh Le
Walmart
Bentonville, AR
USA
minhld38@gmail.com

Young-Woo Kwon
School of Computer Science and Engineering
Kyungpook National University
Daegu, South Korea
ywkwon@knu.ac.kr

Eli Tilevich
Software Innovations Lab
Virginia Tech
Virginia, USA
tilevich@cs.vt.edu

***Abstract*—Edge computing alleviates network bottlenecks by engaging devices at the edge for data processing tasks. These devices possess limited computing resources, while edge execution environments are inherently heterogeneous. It is non-trivial to dynamically allocate the limited resources of heterogeneous devices to balance high performance and low resource utilization. To that end, this paper presents a profiling-based methodology that effectively matches edge computational tasks with the available devices to best satisfy programmer-defined non-functional requirements. Edge tasks are divided into microservices, which are then profiled for their resource utilization. To execute each task, the runtime consults the profiling results to determine the optimal device-to-microservice matching. We realize our methodology as μHTA, whose service-oriented architecture manages the heterogeneity of edge environments and optimally executes microservices on the available mobile devices. We evaluated μHTA by applying it to two realistic edge mobile applications. Our methodology can help developers bridge the gap between the statically specified non-functional requirements and the dynamic nature of edge environments, while reducing the developer burden of optimally utilizing edge-based computing resources.**

***Keywords*—Edge Computing, Profiling, Resource Allocation, Mobile Service Market, Heterogeneity**

I. Introduction

Distributed systems are generating massive amounts of data produced by co-located mobile and IoT devices. The resulting data deluge and its associated bandwidth bottlenecks and privacy concerns often make it infeasible to move all generated data across the network to cloud-based servers for processing. To alleviate this problem, designers of distributed systems are increasingly making use of *edge* and *fog* computing [4]. In these distributed architectures, user-generated data and services operating on it are assigned to nearby edge devices for efficient localized processing and low-latency user access.

If edge processing is to achieve its performance goals, it has to manage its scarce heterogeneous resources effectively [31], [29]. That is, it has to engage the available local computing devices to best match their capabilities with the execution demands of the computing tasks. Such matching is commonly achieved with edge devices interacting with each other directly [11]. Edge environments can differ greatly in terms of their available devices with dissimilar capabilities, a property that we refer to as *edge heterogeneity*. As the Service Oriented Architecture (SOA) has been successfully applied to address the resource and access heterogeneity challenges, computing tasks at the edge are frequently split into services or microservices [30], [21]. Designing applications that take advantage of edge computing resources remains a delicate and complex task [24], [7]. Although the services to perform are known at design time, the edge devices available at runtime can differ across environments. If an edge application needs to perform a fixed set of services, they need to be assigned at runtime to the available best-suited devices [24]. A device's suitability to perform a service is often determined by the necessity to achieve a performance tradeoff (e.g., taking into account processing capability, energy consumption, memory utilization, sensor availability) [7], [31], [42].

To meet the requirements above, developers of edge systems often end up introducing brittle, error-prone, and hard-to-maintain code to the codebase. These problems stand on the way of distributed systems taking full advantage of edge-based resources. Furthermore, to advance the state of the art in edge systems, developers need expressive and robust programming models supported by powerful runtime systems.

Contributions

This paper describes a novel service-oriented methodology that dynamically allocates edge resources in distributed systems to meet developer-defined non-functional requirements (e.g., *minimize* latency, *maximize* memory utilization, *balance* energy consumption, etc.). To that end, for a given edge computing task, our methodology first profiles the task's microservices to determine their resource utilization. Then

[§]Both authors contributed equally to this work.

at runtime, the obtained resource utilization information is consulted to assign the microservices to the available devices, so the developer-specified requirements are best satisfied. By automating this non-trivial decision-making process of matching microservices and available devices, our methodology facilitates the implementation of edge-based applications.

We realize our methodology as *Middleware for Here, There, Anywhere* (or μHTA for short), an edge computing infrastructure for developing and executing distributed systems. μHTA's service-oriented architecture addresses the inherent heterogeneity of edge environments, while its runtime optimally assigns microservices to the best matching available resources. μHTA's cloud-based microservice market effectively deploys code on edge-based devices at runtime. Furthermore, the market model's well-known advantages include vetting the constituent microservices for security and privacy vulnerabilities. We evaluated μHTA by applying it to implement realistic mobile edge applications with promising results.

This paper's contributions include:

- A novel methodology for dynamically allocating edge-based resources; it features a data-driven profiler that correlates performance characteristics of edge-based microservices with available devices, so as to best meet developer-defined non-functional requirements.
- μHTA, an edge computing infrastructure that reifies the methodology above. μHTA includes a programming model, a profiler, and a distributed runtime.
- An empirical evaluation of μHTA that includes applying the infrastructure to implement two realistic mobile edge applications, with promising performance results.

The rest of this paper is structured as follows. In Section II, we present motivating use cases for our work and the technical background required to understand our contributions. Then, we present the design, architecture, and implementation of μHTA in Section III and data-driven profiler used by μHTA in Section IV. In Section V, we benchmark our approach to evaluate its performance during the deployment of two use cases. In Section VI, we discuss μHTA's applicability and limitations. In Section VII, we compare our approach with the related state of the art. Finally, in Section VIII, we outline future work directions and present concluding remarks.

II. Motivation and Background

In this section, we introduce two motivating examples, and then provide the technical background required to understand our approach.

A. Motivating Examples

1) Social Distancing App for COVID-19: Consider developing an app that helps enforce social distancing rules. Assume that an enclosed environment is required to limit the number of simultaneous inhabitants. Users run the app on their mobile and wearable devices. Whenever the social distancing rules are violated, the app warns the user to move to safety, with the warnings delivered context-specifically. For example, at-risk users (e.g., above 30 BMI) are warned differently than other users. Users have to install and configure the app and its services in advance. For example, a user may opt-in to protect her privacy, configuring that certain services be invoked only when a device is within a designated area (e.g., a shopping mall). The main issue of such applications is how to manage the participating devices' mobility, with devices continuously leaving and joining in the computation. Such an application requires that participating devices exchange a lot of data (device information in a short interval), which is very resource-intensive [5].

2) Collaborative Earthquake Detection & Response: Consider developing an app for detecting and responding to earthquakes. Assume that mobile devices in the vicinity collaboratively detect earthquakes and issue an alert with detailed action plans. To monitor ground motion, the app engages a device's accelerometer as a seismic sensor when in steady-state (stationary) [14]. Upon detecting an earthquake-like motion, a device sends the information to participating nearby devices through D2D communication to confirm if it is indeed an earthquake (e.g., increase the model's confidence score). This workflow is tremendously hard to implement and fine-tune, and also very resource-intensive.

B. Technical Background

Next, we introduce the technical background required to be able to understand our contributions.

Microservices: A microservice is a small cohesive piece of functionality or process that can be deployed, scaled, and tested independently [41]. To increase cohesion and reduce coupling, microservices are typically responsible for a single functionality, thus promoting changeability, replaceability, and maintainability [8].

Due to their limited functionalities, microservices typically take a small amount of code to implement. An example of a microservice functionality would be the constituent computations of a numerical model [8]. Microservices can be used to implement highly complex services while offering a high degree of modularity. Developers can easily reuse the smaller parts of their complex services to implement and solve other requirements in different systems. Since all components in a microservices architecture are independent, each component can be tested in isolation. However, integration testing can become very tricky, especially for large systems under test with numerous connections between components [8], [41].

As in any distributed system, security comes to the forefront due to microservices suffering from the same security vulnerabilities as Service Oriented Architecture (SOA) [27], [38]. For example, any time unencrypted microservice data is transferred across the network, it can be intercepted or exfiltrated. Also, due to their emphasis on reuse, microservices commonly integrate third-party functionalities, which may contain their security vulnerabilities [17], [8].

Mobile Service Market: A MSM is a distributed software architecture that delivers services from a centralized repository to connected devices for execution. A variant of SOA, MSM

enables heterogeneous devices to download and execute the services hosted by the marketplace [37], [6].

As edge environments are fundamentally heterogeneous, developers may not determine which specific services will need to execute at runtime. For each service, MSM provides different versions tailored for major mobile platforms. When developing mobile microservices, the resource scarcity of mobile devices must be taken into account. MSM involves interactions across the following stakeholders:

- *Mobile service developers* implement mobile services for all major platforms, so the connected devices can download and execute the services. MSM imposes format and development guidelines. Developers need to follow these guidelines, provide necessary information related to microservices, and upload them to the MSM. The information includes the category of microservices, user manual, sensor requirement, number of device requirements for the microservice implementation. Moreover, it is their responsibility to update and upgrade the new version of microservices.
- *Mobile application developers* use MSM's mobile services in their applications. By browsing through a catalog of MSM services, developers can learn about various characteristics of the available microservices. This knowledge is required to be able to select the services that best meet the given requirements. Developers need to define the prebuilt annotation to select appropriate edge devices for resource execution. Mobile application developers are responsible for designing and developing applications. They need to select the service that resolves the resource scarcity problem at hand and change the constraints of the invoked services for the specific needs of their applications.
- *Mobile users* can opt-in or out for their devices to download and execute microservices of a given MSM. The middleware will manage the service's lifecycle, such as obtaining the latest version of the service execution package for further invocations. Edge application might require specific permission to execute service smoothly; Mobile users should provide the appropriate permission and constraints. Constraints are specific permission of the device's sensor or hardware defined by application developers to maintain user privacy.

An MSM implementation comprises an online repository for hosting microservices, a distributed runtime for managing service lifecycle, and a programming model supporting the development process. Following the application market model makes it possible for mobile users to rely on the reputation of a given market to have enough trust to allow the automatic installation and execution of such mobile services. At the same time, mobile service developers would have to comply with the service market requirements, which likely would have to be more stringent than those of application markets.

III. μHTA OVERVIEW

In this section, we provide an overview of μHTA.

μHTA builds upon the microservice architecture to dynamically allocate the available edge-based resources to best satisfy non-functional requirements. Our design objectives are twofold: (1) bridging the gap between the statically specified non-functional execution requirements and the heterogeneous dynamic nature of edge environments; (2) reducing the developer burden for optimally utilizing heterogeneous edge-based computing resources;

To access edge-based resources efficiently, developers often end up introducing low-level, brittle, and hard-to-maintain code, which is hard and expensive to fine-tune and maintain. μHTA handles the low-level details of distributed interaction by dynamically matching microservices with available computing resources. It also addresses the inherent heterogeneity of edge environments with its service-oriented architecture.

A. General Design

μHTA's primary design principle is to enable developers to integrate ready-made microservices into their applications, thereby reducing programming effort. Hence, our design objective is to retain the software engineering benefits of encapsulation and code reuse and extend them to support efficient resource allocation.

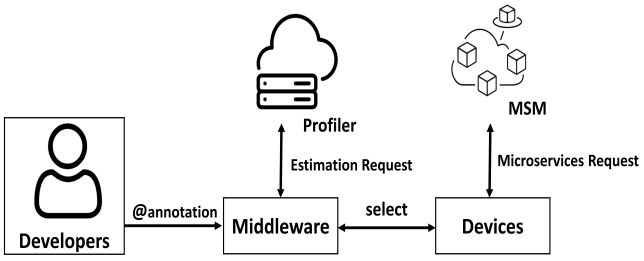

Fig. 1: μHTA's overview.

Figure 1 gives an overview of a μHTA-based system. In it, *developers* provide a set of non-functional requirements in the form of annotations and target microservices. μHTA reads in the annotations provided by the application developer. μHTA collects the device's execution properties and programmer-defined constraints. μHTA requests performance estimation from the profiler based on the available devices' properties. μHTA selects the appropriate devices to execute the target microservices while meeting the developer-defined requirements, thus enabling optimal edge-based resource allocation.

```
1 @Transmission(Minimize=ENERGY,Maximize=LATENCY,
2 Validity=PERMANENT)
3 public Middleware executeMainTest() {...}
```

Fig. 2: Example of annotation usage.

Figure 2 shows a code snippet with an annotation. Application developers annotate methods to express their non-functional requirements. The μHTA's runtime uses this information to determine which devices would be best suited to execute a given microservice. In this case, the annotations

define that when executing this service, the device should aim at `minimizing` the energy consumption irrespective of execution time and that the middleware should not deallocate the microservice after concluding its execution.

B. System Architecture

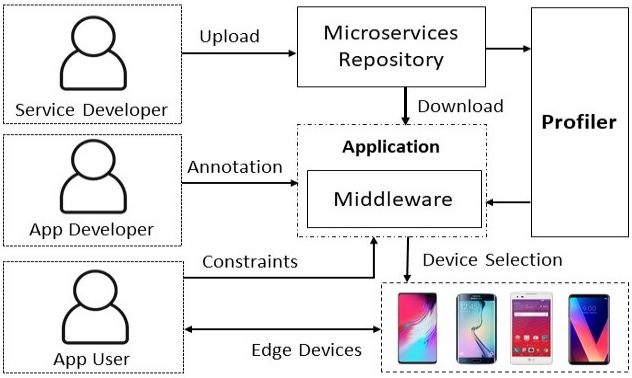

Fig. 3: System architecture.

Figure 3 shows an high-level architecture of a μHTA-based system. In this system, *Edge Devices* refer to the devices that run the application. *Microservice Repository* refers to MSM that contains a set of developer-defined microservices, which the developer previously profiled. The MSM is responsible for managing the deployment of the available microservices while ensuring security and privacy. Finally, the *Profiler* refers to a web application that estimates the resource consumption of a given service or microservice deployment on edge devices.

The *profiler* is responsible for estimating the resource consumption of any runtime edge devices using the data-driven approach. The first step in the data-driven approach is to gather device properties while computing a particular task. These calculated data are preprocessed, filtered, and grouped into different clusters. This information is then analyzed to determine correlations between devices' properties and resource consumption. These correlations are then statically analyzed for estimation. We provide more details of the Profiler in Section IV-A2.

The *middleware* is located in the edge application and acts as the core of μHTA. The middleware automatically downloads and executes microservices. It reads the annotations and device constraints from app developers and app users, respectively. It establishes a peer-to-peer network connection between nearby edge devices. After establishing a network connection, it collects the device's information from participating devices and stores it in the local storage. Since edge devices can send their device properties at any point in time, the middleware only stores the latest information. Moreover, it sends collected device properties information to the profiler, which returns an estimation of the resource utilization. Lastly, it uses the annotations, user constraints, and resource utilization estimations to select which devices are best suited to execute the task.

The middleware also manages data marshaling, local data storage, microservice cache path, resource file storage, and the latest data required to request performance estimation. The middleware also collects the device information, which app developers can extend and integrate with an IDE. The middleware stores each process' data with a unique key; however, it would utilize the latest device information properties when estimating resource consumption. The middleware only allocates resources after the profiling phase has been completed and the application's execution has been initiated.

C. Device Selection Based on Profiling

A profiler located in an edge server provides APIs for edge devices to request resource consumption for either single or multiple edge devices. The primary motivation for separating the profiler from middleware is that the profiler needs a continuous update for real-time data collection. Moreover, it makes the profiler more flexible to include other statistical analyses. Devices need to request via profiler API their device information. The profiler returns an HTTP response in JSON format, which includes the estimated results with devices ordered by priority. Under the successful API calls, device store response results in their local storage.

Algorithm 1 : Device Selection Algorithm

```
1:  Input: edge devices with their properties, Annotation
2:  Ouput: Filter edge devices Dn
3:  function EXECUTETESTFUNCTION(SERVICE_NAME)
4:      download and execute SERVICE_NAME
5:      record and store device resource properties
6:      return save file path
7:  end function
8:  procedure SELECTDEVICE(Dn)
9:      System Initialization
10:     Start Network Connection between Edge Server(Ds) and edge device
11:     Read Annotation
12:     foreach d ∈ Dn do
13:         data = EXECUTETESTFUNCTION(SERVICE_NAME)
14:         Send data to Ds
15:     end foreach
16:     Store Dn information to local storage
17:     Request API to profiler for estimating energy and performance
18:     Filter device based on obtained results and service criteria
19: end procedure
```

Algorithm 1 shows the steps required for selecting the appropriate devices executing μHTA. The algorithm receives as input all the edge devices currently participating in the mobile edge network and the developer-defined annotations. The algorithm's output is a sorted list of devices according to the given annotation and service usage criteria.

Table I shows the different options methods middleware provide to filter and select appropriate devices based on applied annotation. $energy\_min()$ and $energy\_max()$ order devices in ascending and descending order, respectively, concerning energy consumption. Similarly, $performance\_min()$ and $performance\_max()$ order devices based on performance estimation. $lifetime()$ determines the duration of the existence of microservice in the application. Developers can either annotate their services as `TEMPORARY`, which would flag

TABLE I: Devices filtering using annotations.

Annotation	Minimize	Maximize	Permanent
Energy	energy_min()	energy_max()	-
Performance	performance_min()	performance_max()	-
Energy / Performance	avg(energy_min(), performance_min())	avg(performance_max(), energy_max())	-
Validity	-	-	lifetime()

the microservice for deletion upon the user closing the app. Developers can use the `PERMANENT` annotation, which would flag the microservice for storage in the cache directory. If the developer uses multiple annotations, the μHTA calculates the average order.

D. Service execution

```
<?xml version="1.0" encoding="utf-8"?>
<middleware>
    <microservice id="15479647" name="earthquakeService">
        <package_name>com.quake.earthquake</package_name>
        <class_name>EarthquakeService</class_name>
        <main_method>run</main_method>
        <download_attempt_times>3</download_attempt_times>
        <load_from_cache>false</load_from_cache>
        <auth_key>765A795448AE11B2EF9EAAE55FE84</auth_key>
        <method name="run">
            <main_method>true</main_method>
            <parameter>false</parameter>
            <return_data_type>String</return_data_type>
        </method>
         <method name="addInput">
            <parameter>true</parameter>
            <return_data_type>String</return_data_type>
            <number_of_parameter>1</number_of_parameter>
            <first_parameter>String</first_parameter>
        </method>
        </microservice>
</middleware>
```

Listing 1: Service configuration file.

Listing 1 shows the XML configuration file that the middleware uses to download any microservice from the MSM. Application developers are required to download the XML file from the MSM and load it into their application. The MSM automatically generates this file after a developer uploads a microservice and its information (i.e., microservice name, package name, class name, primary method name). The id and name are unique, which separate microservices from each other, while auth_key validates the authenticity of the request. To download the latest version of the microservice, the developer can set the $load\_from\_cache$ key to false and restart the application. Moreover, the developer can also define the number of retry requests that the application performs to download the microservice. The configuration files contain all the methods with their parameter and return types used in the microservices. In this case, there are two method `run` and `addInput` where `run` is the main method that does not take any parameter and return string value. The method `addInput` takes one string parameter and return string output.

E. Failure Handling

The middleware runs on mobile devices that interact with MSM. As all distributed systems, μHTA-based applications are subject to partial failures, such as network disconnection and execution crashes. Our middleware provides fixed failure handling. It maintains heartbeat channels between the connected devices to detect network failures, with a timeout of 10 seconds. If a microservice utilizes external remote resources that are unreachable or its device has insufficient space to run, the middleware retries microservice invocation three times with a 5 seconds pause in between. If the retries are unsuccessful, a failure exception is thrown. To ease debugging, the middleware maintains persistent logs of both failure and success when executing tasks. As a future work direction, we plan to investigate how μHTA can be combined with more flexible and configurable middleware mechanisms for handling failure [16].

IV. Data-Driven Profiling

A. Identifying Resource Usage Patterns

1) Overview: Our objective is to find correlations between the edge-based devices and use them to analyze and compare runtime edge-based devices' energy and performances. Each year tech companies manufacture millions of edge devices with various hardware ranging from different price labels. All these devices' computing capacity might vary from one device to another, and the same device can take various amounts of time and energy to execute the identical task. Moreover, end-user devices setting such as brightness, running background application, and network connectivity also affect the energy and performance. In this paper, we decided to focus on mobile devices as, under the Edge computing umbrella, they constitute one of its significant pillars [2], [25].

Furthermore, there is a vast gap between the energy consumption of mobile devices and IoT devices because of different usage patterns or application types. So, it would be infeasible to compare other types of devices. Also, not all mobile devices support docker images, and the deployment of such would incur additional overhead to the device [40], [35], which we are trying to avoid.

2) Data Collection: For this study, we assigned 25 smartphones to execute the distributed application without changing their settings. These devices executed the application without running any background tasks during the experiments. We collected the following system information that might directly or indirectly affect the devices' performance and energy consumption patterns from the experiments: `device total memory`, `free memory`, `available memory`, `cache memory`, `swap memory`, `app total memory`, `app free memory`, and `app used memory`, `the number of processors`, `maximum frequency`, `current CPU time`, `discharging current` and `voltage`.

At runtime, any device can join or leave the network, and new jobs will be assigned to devices. Thus, the runtime

system needs to be self-sustainable to know about runtime characteristics that are dynamically changed during execution. Moreover, data collection should be efficient in terms of execution time and usages of computing resources.

To fulfill these requirements, we selected a microservice that implements a simple sorting algorithm whose time complexity is $O(N^2)$. We stored non-repeatable 20,000 digits in a CSV file and then ran the experiments in the interval of 30 seconds. In each experiment, a device reads data from the file and sorts them in an array. The process usually takes an average of 10 seconds to complete in a device (e.g., Galaxy S9). In each experiment, a separate thread calculates the device information every 100 milliseconds. The memory, CPU frequency, CPU time, current and voltage information of both a device can be obtained from `/proc/meminfo`, `../cpufreq/scaling_cur_freq`, `/proc/processId/stat`, and Android's BatteryManager class, respectively.

The total number of datasets is($N_{Datasets}$) is:

$$N_{Datasets} = \sum_{i=1}^{d} \sum_{j=1}^{p} (t_{ij} \times 10) \quad (1)$$

Where d is the number of experimented devices (i.e., 25), p is the number of the executed processes (i.e., typically 60), t is the task execution time (i.e., 6 seconds for Galaxy S9), and in every second a device stores ten dataset points.

The energy consumption in a particular process(p) using Ohm's Law [10] is:

$$Energy_p = \frac{\sum_{i=0}^{N} Current_i}{N} \times \frac{\sum_{i=0}^{N} Voltage_i}{N} \times t_p \quad (2)$$

Where N is the number of datasets in each process(p), $Current$ is the discharging current, $Voltage$ is the voltage, and t_p is the execution time of the process p in seconds.

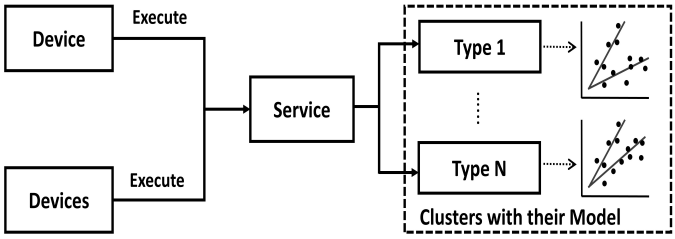

Fig. 4: Clustering process.

3) Clustering Collected Data: Figure 4 shows the workflow diagram of the clustering process that occurs inside the profiler. A study conducted by a Dutch non-profit organization, Consumentenbond estimates that the average lifespan of mobile devices is 2.5 years [1], and the high-end devices of the past few years are today's low-end devices. Moreover, it is practically impossible to pre-profile services for all devices. We also found considerable energy and performance gaps of the same devices in our collected data, so instead of classifying devices based on the device name, we cluster them based on their properties, i.e., the same device data might belong to different types. Such a cluster's main advantages are that the runtime device would belong to the cluster depending on its resource allocation.

The number of clusters is changeable and might change in the future depending upon the data collected. After merging each process and applying data preprocessing to current datasets, we had a total of 7,003 data points. To estimate the number of clusters, we use the Elbow method [12] and the Silhouette method [32]. The Elbow method focus more on decision rules, while the Silhouette is a metric used for validation while clustering.

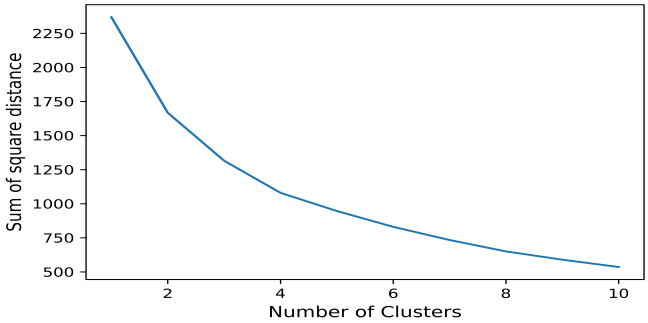

Fig. 5: Elbow method for optimal cluster

Figure 5 shows the elbow method for different clusters; in the elbow method, the cluster at the "elbow" decreasing in a linear format is considered an optimal number of clusters. The elbow is ambiguous for clusters two, three, four, and five; hence to choose the appropriate cluster, we calculate the silhouette value that shows the data point's similarity with its cluster compared to other clusters.

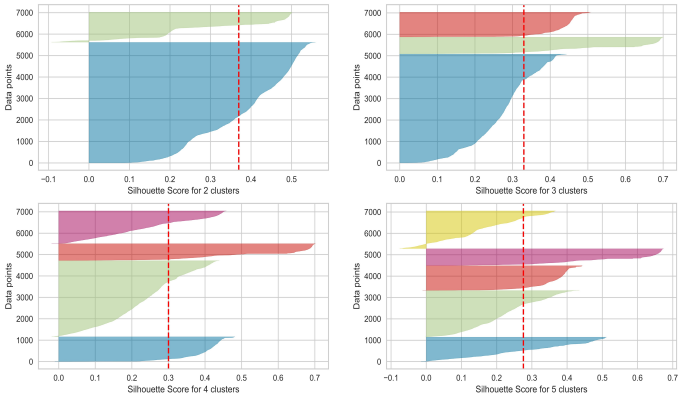

Fig. 6: Silhouette analysis for 2, 3, 4, and 5 clusters.

Figure 6 shows the silhouette analysis for two, three, four, and five clusters. The silhouette score for all the clusters is comparatively similar to each other. The two and three clusters have more than 80% and 63% data belonging to a single cluster, so the cluster looks sub-optimal for the given dataset. The silhouette score of four clusters is more significant than

five clusters. However, the size distribution of five clusters is comparatively uniform than four clusters. Thus, we analyzed the regression model's accuracy of four and five clusters and found that four clusters are more relevant than five clusters. As a result, we selected four clusters with 782, 3,547, 1,152, and 1,522 data points.

4) Threats to Validity: Because all devices' versions used in this experiment are greater than Android version 6 (i.e., Marshmallow), our method may not cluster low-end devices with Android API Level less than 23.

B. Data Analysis for Profiler

1) Motivation and Research Question: With the heterogeneity and scale of edge-based systems, designers may find it overwhelming to fulfill all their requirements. One of the hardest system design issues of the edge is effectively and efficiently dynamically allocating resources. To make inroads in tackling this issue, we analyze the collected data for the presence of correlations between the performance metrics, type of computing task, and device type, thus seeking answers to the following question: `MQ1` – which performance metrics are correlated to task-type?

2) Methodology: To answer `MQ1`, we clustered the data collected in Section IV into four groups. We then applied polynomial regression models to identify whether the collected performance metrics are correlated to energy consumption. We used coefficient determination (R-2), F-test, and root mean squared error (RMSE) to determine the accuracy of the regression model. R-2 is a statistical measure of how close the data are to the regression line. F-test checks if the variances of two populations are the same by comparing the ratio of the variances. An F-test of 1 indicates that two populations are the same. The RMSE represents the differences between estimated and observed values. Additionally, it measures the model's accuracy. We used the measurements gathered from the prior section for the regression model.

In total, we used $7,003$ data points from the empirical study. For the regression model, we divided the data sets into two parts, $80\%(5,602)$ of the data set was used for the training model and the remaining `20% (1,401)` for testing. We adhere to a higher model accuracy than prior research on modeling performance in distributed applications [22]. A regression model is considered successful only if its accuracy exceeds 60%.

3) MQ1: On Task Type and Energy Consumption: We followed the methodology described above and applied polynomial regression to collected metrics. By doing so, we observed that the energy consumption of `type 0` and `type 1` tasks are correlated to `cpu time`, `current`, `voltage`, `execution time`, and `device type`. We observed the following indicators for the multilinear regression models for `type 0 tasks:` R^2 0.99606 MSE $126,162,539.47$; `type 1 tasks:` R^2 0.998057 & MSE $137,792.296$. Also, we observed that the energy consumption of `type 2` tasks are correlated to `cpu time`, `current`, `voltage`, and `execution time`. We observed the following indicators for the multilinear regression models for `type 2 tasks:` R^2 0.873107 MSE $547,033,728.69271$; `type 1 tasks:` R^2 0.998057 & MSE $137,792.296$.

We observed that the energy consumption of `type 3` tasks are correlated to `current`, `voltage`, `execution time`, and `device type`. We observed the following indicators for the multilinear regression models for `type 3 tasks:` R^2 0.715 MSE $141,361,947.28$.

4) Summary of the Results: We developed three regression models by leveraging the t-statistic data obtained from regression analysis, it was determined if the dependent variables are influential in describing the variation of the dependent variable. The accuracy of the regression models was determined using coefficient determination (R^2), F-test, and root mean squared error (RMSE). The developed regression models for energy consumption are available here: https://github.com/uhta524/uhta.

V. Evaluation

For the evaluation, we had two benchmarks and two case studies with realistic scenarios.

A. Micro Benchmark

For the evaluation, we deployed μHTA on Samsung S9, LG710, Galaxy S7, Galaxy S10, and LG300 with the target OS Android 10. For power measurements, we used the device's internal function to gather the energy consumption information. Our evaluation objectives are to evaluate the differences while implementing μHTA with random device selection in terms of their respective software engineering metrics runtime performance (i.e., time and energy consumption).

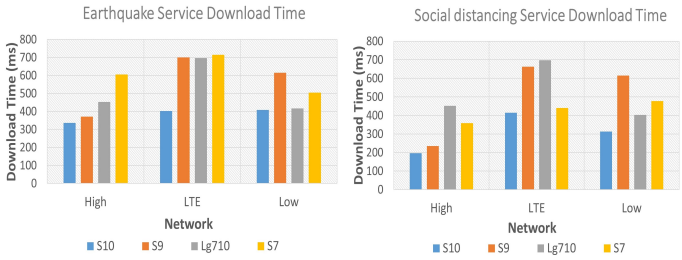

Fig. 7: Microservice download time in ms.

Figure 7 shows the time middleware takes to download a microservice for our case study of earthquake detection and social distancing, each of size 8.2Kb and 9.94 Kb, respectively. We used the high-frequency network of 5GHz (High), low-frequency network of 2.4GHz (Low), and 4G LTE network (LTE) to download service on four experimental devices. To measure Internet speed, we use fast.com[§] before each experiment. The average speed of the network was 130—150 Mbps, 21—26 Mbps, and 10—18 Mbps for High, LTE, and Low networks, respectively. The devices take an average of

[§]Fast.com helps to check the internet speed people are getting from their internet service provider.

300 to 500 milliseconds and 200 to 700 milliseconds to download and load earthquake and social distancing microservices, respectively. Once the devices download the microservice, they can load it from the cache directory, usually taking around 4 to 10 milliseconds.

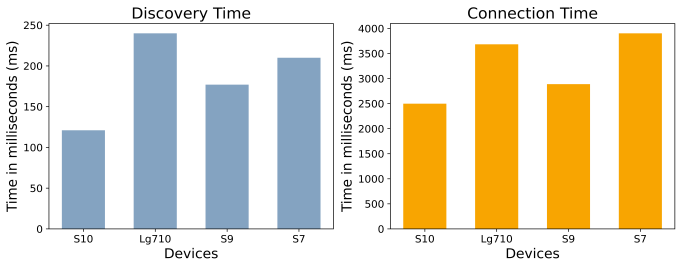

Fig. 8: Network connection time in ms.

Figure 8 shows the time to discover and connect to the edge devices. The average time to discover any device is between 120 to 240 milliseconds and to connect with the device is between 2,500 to 3,900 milliseconds. We also found that the most high-end device takes less time to discover and connect than other devices.

B. Case Study

Figure 9 shows the architecture diagram implementing microservice using μHTA. D1, D2, D3, and D4 are the edge devices participating in the network. Each device executes the test function and shares the device resource information with the edge server that selects appropriate service execution devices.

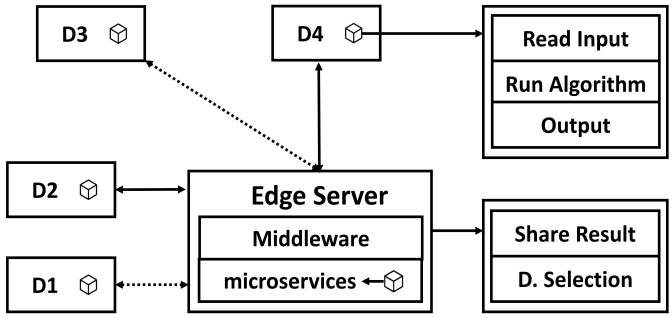

Fig. 9: Use case architecture.

1) Earthquake Detection: In collaborative earthquake detection, the earthquake microservice includes a sensor reader, a trigger estimator, a feature extractor, an earthquake prediction model, and an alert notification algorithm. This microservice monitors the vibrations using an accelerometer. When a given threshold is exceeded, the microservice analyzes two seconds of prior data to extract features required to predict whether an earthquake is in progress. If the earthquake probability exceeds 90%, the participating devices forward it to the edge server and then await its response. In the meantime, the edge server collects the probability reported by each participating device and then calculates the majority by voting. If the calculated final result indicates that an earthquake is indeed in progress, then the service notifies the participating devices by sending them individualized response plans.

TABLE II: Earthquake service resource consumption.

Device	Avg(Execution Time (ms))	Avg(Energy Consumption (J))
S10	5,729	7,488
S9	5,788	7,582
S7	6,574	8,954
LG300	5,294	8,263
LG710	5,603	6,077

Table II shows the average resource consumption while executing the earthquake microservice in μHTA. To estimate the average execution time (ms) in milliseconds and energy in Joules, we executed the application in five different smartphones 100 times. In each process, we measured the execution time from when a smartphone detects a trigger to when an alert notification is displayed on the smartphone. We calculated each execution's energy consumption by multiplying the average discharging current, average voltage, and execution time in seconds.

2) Social distancing detection: The microservice that detects social distancing first retrieves the current location, then collects the users' health information, and finally alerts the appropriate users, as guided by the notification algorithm. The edge devices share the device properties, its location, and the user-health information every 10 seconds. The edge server calculates the active number of devices and creates an alert notification once the device exceeds a certain threshold. The developer provides an interface to collect users' health information.

C. Evaluation Comparison

To evaluate μHTA, we compare the result with Device Resource Level (DRL), representing the execution capability of a mobile device proposed in [18], [19] papers. DRL considers the CPU information and estimates the available computing resource capacity of edge devices. It compares the CPU resource utilization of edge devices based on a mathematical model and tries to select optimal devices.

TABLE III: Resource consumption of the test service.

Devices	Avg(DRL)	Avg(Time)	Avg(Predicted Energy (J))
S10	6,932,013	6,163	31,979
S9	3,456,934	6,107	23,283
S7	3,517,083	10,681	41,222
LG300	3,528,067	23,288	32,146
LG710	3,485,300	5,380	14,899

Table III shows the average value of DRL, Average Execution Time, and average Predicted Energy consumption using

μHTA in 50 experiments. The test service is the same; bubble sort service used while collecting data.

TABLE IV: Estimation of device rankings.

Devices	Avg(MER)	Avg(MPR)	Avg(MEMPR)	Avg(MDRLR)
S10	4	2	3	1
S9	2	3	2	5
S7	5	4	5	3
LG300	3	5	4	2
LG710	1	1	1	4

Table IV shows the average ranking of devices based on the estimated energy, performance, and DRL in 50 experiments. MER, MPR, MEMPR, MRLR represents Minimum Energy Ranking, Minimum Performance Ranking, Minimum Energy and Maximum Ranking, and Maximum Device Resource Level Ranking, respectively. The ranking is the order of devices from 1 to 5 based on the calculated value.

In most of our experiments, we found that DRL was selecting the same devices; however, device selection was different using our middleware. DRL uses the same equations and depends upon few properties of devices for device selection. μHTA runtime balances the developer's non-functional requirements with the changing device's properties to select based on the best-suited execution environment. Our middleware provides more flexible options for the developer to select devices running in the edge environment in terms of Maximize, Minimize resource utilization compared to DRL.

VI. Applicability & Limitations

In this section, we present guidelines for edge application developers, with the purpose of helping them successfully apply μHTA in their development practices. We also discuss the limitations of our approach.

A. Usage Guidelines

μHTA efficiently and effectively allocates edge resources to execute services while following developer-defined non-functional requirements. μHTA maintains a list of the most recent resource-utilization profiles of the edge-services and the participating devices. Furthermore, μHTA reads in the developer-defined annotations, which specify the desired non-functional requirements for the runtime (e.g., low-energy consumption, low-latency, or a balance in terms of resource-utilization). By collecting and managing all the available information about both the microservices and the available devices, μHTA effectively manage the complexity of optimally allocating distributed edge resources.

For example, in the *collaborative earthquake prediction* use-case, developers are relieved from having to profile and manage the execution of the earthquake detection service. Also, developers are not required to optimize the resource allocation to execute the services, all handled by the μHTA's runtime.

B. Limitations

μHTA makes decisions by following a data-driven approach that has some inherent drawbacks. To minimize these drawbacks, μHTA collects consistent data from different devices, accurately pre-processes the data, clusters it, and applies a high accuracy prediction model. However, μHTA's model needs continuous updates on newly released devices for high accuracy. The performance of edge applications is directly dependant on the available network's capacities and participating devices. The issues of privacy are left as out of scope. However, edge-based applications already offer privacy superior to those of cloud applications, with the raw sensor data never leaving the user devices.

VII. Related Work

The work presented here is related to other complementary efforts that improve the reliability and efficiency of mobile distributed execution, including frameworks, peer-to-peer networking, code migration, and computation offloading.

In mobile resource scheduling and allocation, Nawrocki [26] introduces an adaptive task scheduling system that optimizes mobile devices' energy consumption using machine learning mechanisms to learn how to allocate resources appropriately by scheduling services/tasks optimally between the device and the cloud. Similarly, MALMOS [9], a framework for mobile offloading scheduling based on online machine learning, provides an online training method for the runtime scheduler, so it supports a flexible policy that dynamically adapts scheduling decisions based on the observation of previous offloading decisions and their correctness. To avoid trunk of data being transferred during the pre-optimization process, Z. Ali et al. [3] develop an energy-efficient deep learning-based offloading scheme (EEDOS), with the core smart decision-making algorithm based on deep learning to optimize the offloading components based on a number of inputs such as remaining energy of user devices, energy consumption by app components, network latency, and failures, computational load and amount of data. They developed the cost function based on these inputs and selected the optimal policies over an exhaustive dataset to train a deep learning network instead of examining actual executions. Yunzhao Li et al. [20] develop a deep reinforcement learning algorithm to handle the offloading tasks for the heterogeneous Edge Computing Server(ECS) collaborative computing. This approach is based on the real-time state of the network and the task properties, using Actor Critic and Policy Gradient's Deep Deterministic Policy Gradient (DDPG) to optimize computation decisions offloading. F. Sufyan et al. [39] propose an efficient computation offloading scheme for a distributed load sharing edge computing network in cooperation with cloud computing to enhance the capabilities of the mobile devices. They formulate a nonlinear multiobjective optimization problem by using queuing theory to model the execution delay, energy consumption, and payment cost for using edge and cloud services. They propose a stochastic gradient descent (SGD) algorithm-based solution approach to optimize the offloading probability and transmission power of

the mobile devices to find an optimal trade-off between energy consumption, execution delay, and cost of the devices.

Weisong et al. [36] summarized the visions and challenges of edge computing with multiple case studies, ranging from cloud offloading to smart home and city, collaborative edge to materialize the concept of edge computing. In the world of IoT devices, Edge Mesh [34] considers low-level devices by distributing the decision-making tasks among edge devices within the network instead of sending all the data to a centralized server. All the computation tasks and data are shared using a mesh network of edge devices and routers. Yuvraj et al. [33] study data-aware task allocation problem to jointly schedule task and network flows in collaborative edge computing and mathematically model the joint problem to minimize the overall completion time of the application. They proposed a multistage greedy adjustment (MSGA) algorithm where the task scheduling is done by considering both placement of tasks and adjustment of network flows. Through simulation, the MSGA algorithm improved at least 27% compare with the other bench-marked solutions. To support IoT collaboration in cooperative vehicle-infrastructure system (CVIS), J. Zhou et al. [43] develop an analytical framework of reliability-oriented cooperative computation optimization, considering the dynamics of vehicular communication and computation. In particularly, they applied stochastic modeling of V2V and V2I communications, take into account the effects of the vehicle mobility, channel contentions, fading, and theoretically derive the probability of successful data transmission. These approaches lack an efficient programming model and a runtime infrastructure for developers/experts to embed the middleware and components into their applications in the easiest way without changing its architecture.

Regarding profiling in edge computing, Woo-Joong et al. [15] introduce an online profiling-based configuration adaptation for video analytics systems by analyzing frame rate and resolution to optimize resource accuracy tradeoffs of multiple video streams on edge servers. Xinchen et al. [23] design a new selective offloading mechanism that employs a program profiler and decision engine at the client side to make an offloading decision, which is to reduce workload for edge servers; this approach claims to optimize the performance of the Green IoT. Similarly, DeepDecision [28] is a framework to optimize the execution of deep learning on cloud/edge servers to support mobile devices. They consider model accuracy, video quality, battery constraints, network data usage, and network conditions to determine an optimal offloading strategy and bring deep learning to mobile devices in an effective way. Hermes [13] utilizes the NP-hard problem to minimize the latency of application and meet device resource constraints; it uses a fully polynomial-time approximation scheme to ensure theoretical performance. This approach adapts to the unknown dynamic environment and guarantees the performance gap compared to the optimal strategy by a logarithmic function with time.

VIII. CONCLUSIONS AND FUTURE WORK

For future research, we have the following research directions. First, we will increase the data size by including IoT devices or more smartphones. Currently, our model used for the profiler is based on a straightforward multi-regression technique, so we will explore advanced machine learning approaches using deep neural networks. Moreover, we will provide a plug-in that can integrate with modern IDEs for software developers who are not familiar with edge computing. We will extend our microservice-based architecture to a Docker-based service architecture. Finally, we will adopt energy-efficient technologies like low Bluetooth energy and Neighbor Awareness Networking (NAN) for network connectivity.

In this paper, we presented μHTA, a profiling-based service platform for dynamically allocating edge resources to maximize energy consumption, execution latency, and memory utilization objectives. μHTA divides a computation task into a set of services and profiles their resource usage patterns. At runtime, μHTA assigns the corresponding services to appropriate devices that meet a developer's non-functional requirements. Because μHTA is based on a service-oriented architecture, all the deployed services through our MSM can be profiled by μHTA. For the evaluation, we measured the network performance of our runtime system and then implemented two use cases using our μHTA— collaborative earthquake detection and social distancing. We found that μHTA provides more flexible alternatives to software developers.

ACKNOWLEDGEMENTS

The authors would like thank the anonymous reviewers, whose insightful comments helped improve this paper. This research is supported by the National Research Foundation of Korea (NRF) grant funded by the Korea government (MSIT) (No.2021R1A5A1021944) and by Basic Science Research Program through the NRF funded by the Ministry of Education (NRF-2021R1I1A3043889). This research is also supported by US NSF through the grant #1717065. Y. Kwon and E. Tilevich are co-corresponding authors.

REFERENCES

[1] JESPER, What is the lifespan of a smartphone . https://www.coolblue.nl/en/advice/lifespan-smartphone.html, 2021. Last accessed 20 July 2021.

[2] What are Edge Computing Devices? https://www.exorint.com/en/blog/what-are-edge-computing-devices, 2021. Last accessed 20 July 2021.

[3] Z. Ali, L. Jiao, T. Baker, G. Abbas, Z. H. Abbas, and S. Khaf. A deep learning approach for energy efficient computational offloading in mobile edge computing. *IEEE Access*, 7:149623–149633, 2019.

[4] F. Bonomi. Connected vehicles, the internet of things, and fog computing. In *The Eighth ACM International Workshop on Vehicular Inter-Networking (VANET), Las Vegas, USA*, pages 13–15, 2011.

[5] S. Brienza, S. E. Cebeci, S. S. Masoumzadeh, H. Hlavacs, Ö. Özkasap, and G. Anastasi. A survey on energy efficiency in p2p systems: File distribution, content streaming, and epidemics. *ACM Computing Surveys (CSUR)*, 48(3):1–37, 2015.

[6] B. D. Cruz, J. Cheng, Z. Song, and E. Tilevich. Understanding the potential of edge-based participatory sensing: an experimental study. In *2020 IEEE 91st Vehicular Technology Conference (VTC2020-Spring)*, pages 1–5. IEEE, 2020.

[7] B. D. Cruz, A. K. Paul, Z. Song, and E. Tilevich. Stargazer: A deep learning approach for estimating the performance of edge-based clustering applications. In *2020 IEEE International Conference on Smart Data Services (SMDS)*, pages 9–17. IEEE, 2020.

[8] N. Dragoni, S. Giallorenzo, A. L. Lafuente, M. Mazzara, F. Montesi, R. Mustafin, and L. Safina. Microservices: yesterday, today, and tomorrow. In *Present and ulterior software engineering*, pages 195–216. Springer, 2017.

[9] H. Eom, R. Figueiredo, H. Cai, Y. Zhang, and G. Huang. Malmos: Machine learning-based mobile offloading scheduler with online training. In *2015 3rd IEEE International Conference on Mobile Cloud Computing, Services, and Engineering*, pages 51–60, 2015.

[10] M. S. Gupta. Georg simon ohm and ohm's law. *IEEE Transactions on Education*, 23(3):156–162, 1980.

[11] Y. C. Hu, M. Patel, D. Sabella, N. Sprecher, and V. Young. Mobile edge computing—a key technology towards 5g. *ETSI White Paper*, 11(11):1–16, 2015.

[12] K. D. Joshi and P. Nalwade. Modified k-means for better initial cluster centres. *International Journal of Computer Science and Mobile Computing*, 2(7):219–223, 2013.

[13] Y. Kao, B. Krishnamachari, M. Ra, and F. Bai. Hermes: Latency optimal task assignment for resource-constrained mobile computing. *IEEE Transactions on Mobile Computing*, 16(11):3056–3069, 2017.

[14] I. Khan, M. Pandey, and Y.-W. Kwon. An earthquake alert system based on a collaborative approach using smart devices. In *2021 IEEE/ACM 8th International Conference on Mobile Software Engineering and Systems (MobileSoft)*, pages 61–64, 2021.

[15] W. Kim and C. Youn. Lightweight online profiling-based configuration adaptation for video analytics system in edge computing. *IEEE Access*, 8:116881–116899, 2020.

[16] Y.-W. Kwon, E. Tilevich, and T. Apiwattanapong. DR-OSGi: Hardening distributed components with network volatility resiliency. In *ACM/IFIP/USENIX Middleware 2009*.

[17] X. Larrucea, I. Santamaria, R. Colomo-Palacios, and C. Ebert. Microservices. *IEEE Software*, 35(3):96–100, 2018.

[18] M. Le and Y.-W. Kwon. Efficiently sharing remote computing resources for mobile devices. *IEIE Transactions on Smart Processing & Computing*, 9(4):336–343, 2020.

[19] M. Le, M. Song, and Y. Kwon. Enabling flexible and efficient remote execution in opportunistic networks through message-oriented middleware. In *2017 IEEE 41st Annual Computer Software and Applications Conference (COMPSAC)*, volume 1, pages 979–984, 2017.

[20] Y. Li, F. Qi, Z. Wang, X. Yu, and S. Shao. Distributed edge computing offloading algorithm based on deep reinforcement learning. *IEEE Access*, 8:85204–85215, 2020.

[21] C.-F. Liu, M. Bennis, M. Debbah, and H. V. Poor. Dynamic task offloading and resource allocation for ultra-reliable low-latency edge computing. *IEEE Transactions on Communications*, 67(6):4132–4150, 2019.

[22] W. Lloyd, S. Pallickara, O. David, J. Lyon, M. Arabi, and K. Rojas. Performance modeling to support multi-tier application deployment to infrastructure-as-a-service clouds. In *2012 IEEE Fifth International Conference on Utility and Cloud Computing*, pages 73–80. IEEE, 2012.

[23] X. Lyu, H. Tian, L. Jiang, A. Vinel, S. Maharjan, S. Gjessing, and Y. Zhang. Selective offloading in mobile edge computing for the green internet of things. *IEEE Network*, 32(1):54–60, 2018.

[24] P. Mach and Z. Becvar. Mobile edge computing: A survey on architecture and computation offloading. *IEEE Communications Surveys & Tutorials*, 19(3):1628–1656, 2017.

[25] C. Mahmoudi, F. Mourlin, and A. Battou. Formal definition of edge computing: An emphasis on mobile cloud and iot composition. In *2018 Third international conference on fog and mobile edge computing (FMEC)*, pages 34–42. IEEE, 2018.

[26] P. Nawrocki and B. Sniezynski. Adaptive context-aware energy optimization for services on mobile devices with use of machine learning. In *Wireless Personal Communications 2020*. IEEE, 2020.

[27] R. Perrey and M. Lycett. Service-oriented architecture. In *2003 Symposium on Applications and the Internet Workshops, 2003. Proceedings.*, pages 116–119. IEEE, 2003.

[28] X. Ran, H. Chen, X. Zhu, Z. Liu, and J. Chen. Deepdecision: A mobile deep learning framework for edge video analytics. In *IEEE INFOCOM 2018 - IEEE Conference on Computer Communications*, pages 1421–1429, 2018.

[29] J. Ren, G. Yu, Y. He, and G. Y. Li. Collaborative cloud and edge computing for latency minimization. *IEEE Transactions on Vehicular Technology*, 68(5):5031–5044, 2019.

[30] Y. Ren, Z. Weng, Y. Li, Z. Xie, K. Song, and X. Sun. Distributed task splitting and offloading in mobile edge computing. In *International Conference on Communications and Networking in China*, pages 33–42. Springer, 2019.

[31] J. M. Rodriguez, C. Mateos, and A. Zunino. Energy-efficient job stealing for cpu-intensive processing in mobile devices. *Computing*, 96(2):87–117, 2014.

[32] P. J. Rousseeuw. Silhouettes: a graphical aid to the interpretation and validation of cluster analysis. *Journal of computational and applied mathematics*, 20:53–65, 1987.

[33] Y. Sahni, J. Cao, and L. Yang. Data-aware task allocation for achieving low latency in collaborative edge computing. *IEEE Internet of Things Journal*, 6(2):3512–3524, 2019.

[34] Y. Sahni, J. Cao, S. Zhang, and L. Yang. Edge mesh: A new paradigm to enable distributed intelligence in internet of things. *IEEE Access*, 5:16441–16458, 2017.

[35] E. A. Santos, C. McLean, C. Solinas, and A. Hindle. How does docker affect energy consumption? evaluating workloads in and out of docker containers. *Journal of Systems and Software*, 146:14–25, 2018.

[36] W. Shi, J. Cao, Q. Zhang, Y. Li, and L. Xu. Edge computing: Vision and challenges. *IEEE Internet of Things Journal*, 3(5):637–646, 2016.

[37] Z. Song, M. Le, Y.-W. Kwon, and E. Tilevich. Extemporaneous micro-mobile service execution without code sharing. In *2017 IEEE 37th International Conference on Distributed Computing Systems Workshops (ICDCSW)*, pages 181–186. IEEE, 2017.

[38] D. Sprott and L. Wilkes. Understanding service-oriented architecture. *The Architecture Journal*, 1(1):10–17, 2004.

[39] F. Sufyan and A. Banerjee. Computation offloading for distributed mobile edge computing network: A multiobjective approach. *IEEE Access*, 8:149915–149930, 2020.

[40] S. S. Tadesse, F. Malandrino, and C.-F. Chiasserini. Energy consumption measurements in docker. In *2017 IEEE 41st Annual Computer Software and Applications Conference (COMPSAC)*, volume 2, pages 272–273. IEEE, 2017.

[41] J. Thönes. Microservices. *IEEE software*, 32(1):116–116, 2015.

[42] A. Toor, S. ul Islam, G. Ahmed, S. Jabbar, S. Khalid, and A. M. Sharif. Energy efficient edge-of-things. *EURASIP Journal on Wireless Communications and Networking*, 2019(1):1–11, 2019.

[43] J. Zhou, D. Tian, Y. Wang, Z. Sheng, X. Duan, and V. C. M. Leung. Reliability-optimal cooperative communication and computing in connected vehicle systems. *IEEE Transactions on Mobile Computing*, 19(5):1216–1232, 2020.