2018 IEEE/ACM International Conference on Utility and Cloud Computing Companion (UCC Companion)

A Smart Testing Framework for IoT Applications

Brian, Ramprasad
York University
brianr@yorku.ca

Joydeep, Mukherjee
York University
jmukherj@yorku.ca

Marin, Litoiu
York University
mlitoiu@yorku.ca

***Abstract*—The number of IoT devices has been growing exponentially as new products are developed and legacy systems become Internet enabled. As a consequence, the large amount of traffic generated by IoT devices require new approaches to network architecture design. The primary challenge with IoT devices is that the traffic can be highly variable due to the device type and the time of use. In order to maintain Quality of Service standards in a dynamic IoT network, these traffic patterns need to be modeled and understood so that we can adapt the architecture dynamically to maintain a desired level of Quality of Service. To this effect, we propose a Smart Testing Framework that can detect bottlenecks and predict the demand for computing resources in a dynamic IoT network. Results obtained from using our framework indicate that we can predict the demand for computing resources in a dynamic IoT network with a high degree of accuracy.**

I. INTRODUCTION

With a growing increase in the volume and variety of connected IoT devices, the demand on the supporting IoT network infrastructure is constantly changing. IoT applications that process data from the connected IoT devices require an IoT network architecture that can handle a huge bulk of data from a large number of IoT devices. IoT networks developed currently are typically heterogeneous in nature, i.e., there are many different types of IoT devices that behave differently which are connected to the network. This is in contrast to homogeneous IoT networks where all IoT devices are identical.

One of the key challenges with heterogeneous IoT networks is maintaining the quality of service (QoS) because of the fluctuations in the number of active devices and the data they produce [1], [2], [3]. Fluctuations occur because IoT devices may operate under different time of use policies to save energy, IoT devices may fail in the network or their up link to the Internet may be temporarily down. Being able to reliably predict resource utilization in a dynamic heterogeneous IoT environment can help overcome some of the QoS challenges associated with scaling IoT networks. Prediction with high accuracy allows us to plan ahead in preparing for changes in demand on the IoT network. For e.g., some critical IoT applications that are used in the medical field require that the data be processed with minimal latency [4]. If the resource utilization in the supporting IoT network becomes too high due to an increase in the number of IoT devices, this may impact the time to process the data by the application. Hence, the QoS for such IoT applications can degrade and this can have serious consequences in medical systems.

Our work is focused on understanding the behavior of highly dynamic and heterogeneous IoT networks. Specifically, we aim to detect the resource bottleneck in a heterogeneous IoT network. Furthermore, we also focus on predicting the resource utilization in an IoT network when more IoT devices are added to it. This is important since the lack of knowledge regarding bottlenecks in IoT networks can cause the QoS of IoT applications to degrade. As described before, QoS is important for certain types of IoT applications that require fast processing where end users need fast answers to queries. For example, if it has been agreed that a query on a set of IoT data should take no longer than 3 seconds to complete, then we should be able to predict at what resource utilization level the query takes longer than 3 seconds to complete. We aim to achieve these objectives by answering the following research question: Can we identify a bottleneck and trigger an adaptive action by using a prediction algorithm before a service level violation occurs in a heterogeneous IoT network?

Identifying bottlenecks in IoT networks is a challenging problem since the devices that are present on a heterogeneous IoT network are constantly subjected to change. Since the volume and variety of IoT devices on a heterogeneous network can be dynamic, it becomes a challenge to predict the need for computing resources. Therefore an investigation and comparison of bottleneck prediction methods is an important process that IoT network architects must undertake before one can be confident that the system will perform according to established QoS levels.

In this paper, we develop a novel resource utilization prediction engine for IoT applications based on a Smart Testing Framework for Adaptation. This allows us to execute repeatable experiments to learn about IoT device resource utilization so that we can trigger adaptations to add or remove computing resources. The framework also allows for new prediction modules to be implemented including learning models through a systematic collection of historical data and analysis. For example, we can set the data collection feature of the framework to continuously monitor the IoT device count and type on the network and the corresponding system resource utilization. We can then store this information in a persistent database in order to generate a dataset. This dataset can then be used to train a model that can predict resource utilization in an IoT network.

The smart testing framework is implemented on the Emulated Internet of Things or (EMU-IoT) platform developed at York University. EMU-IoT is a containerized IoT emulation environment where software defined IoT devices can be instantiated at scale to simulate near realistic traffic on IoT networks. We modeled heterogeneous IoT networks on

978-1-7281-0359-4/18/$31.00 ©2018 IEEE
DOI 10.1109/UCC-Companion.2018.00064

this platform where we had a combination of devices that produce small and large amounts of data. We executed two types of experiments. First, we executed experiments using an exhaustive search to determine the number of IoT devices to reach a target resource utilization. This data was then used as input for a linear regression prediction model to find the device count for a unknown CPU utilization target. We were able to show that predicting the usage patterns of light weight IoT Devices can be done with linear regression models with a high degree of accuracy.

II. Smart Testing Framework

In this section we describe the various components of our proposed Smart Testing Framework and how it will help us to achieve our research goals of being able to intelligently trigger scaling adaptations in highly dynamic heterogeneous IoT networks.

A. Test Cases

Test cases are designed to allow us to model and investigate the behavior of an IoT application in response to a particular scenario on the IoT network. For example, in a smart building we may want to activate IoT environmental control devices during the day and shut them down at night. So, to emulate this scenario we would define the rules to generate a test case that creates IoT devices at certain times of the day. There are several scenarios in which we emulate these test cases: Geographical Distribution, Temporal Distribution, Heterogeneity, Network Connectivity Variety, Network Protocol Variety, and Infinitely Scalable Design. For example, in Fig. 1 we model the scenario where we create IoT Devices on until we reach a CPU resource utilization target and then signal a scaling action.

```
while CPU Utilization <= "80%"
  Create "n" of IoT Device Type "A", every "5"
      seconds
  Create "n" of IoT Device Type "B", every "5"
      seconds
  if CPU Utilization in Application "A" > "80%"
    Trigger scaling action to add more
        resources.
```

Fig. 1. Rules for Scaling

B. State Machine

The primary component of the Smart Testing Framework is the state machine, as shown in Fig. 2. The state machine controls the process of creating workloads and detecting bottlenecks in the IoT network. There are three distinct states that the machine can be in until it exits.

1) ***Generate Test Case***
   Based on the type and configuration of a test case, a new case is created each time this state is reached. This means that a new set of IoT devices is created or removed and there will be a resulting change in the traffic on the network that can be observed and monitored. The test case can be generated based on a set of predefined rules or a set of rules generated by the prediction engine that will be discussed later in Section II-C.

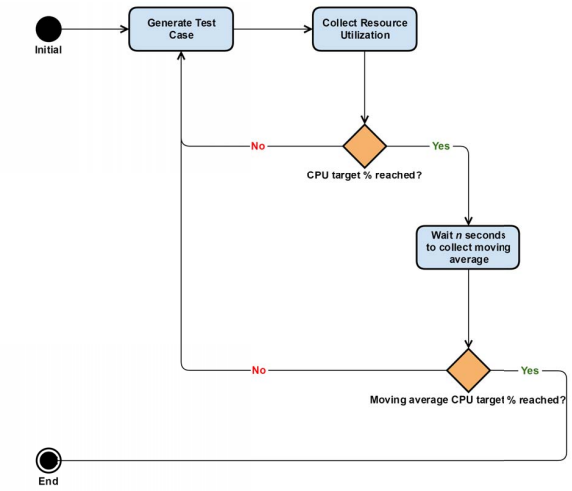

Fig. 2. Smart Testing State Machine

2) ***Collect Resource Utilization***
   In this state, the resource utilization is checked in real time to see if the defined target utilization has been reached when more IoT devices are added to the IoT network. The resource utilization information is polled from the server periodically and checked to determine if it is greater than or equal to the target utilization.
3) ***Collect Moving Average***
   The purpose of this state is to temporarily pause the generation of new test cases to observe the resource utilization under more stable conditions. This is because after the test case has created new IoT devices they may require some time to initialize and begin generating traffic. This state is also important to eliminate random spikes in resource utilizations in the IoT network due to application background processes that are unrelated to the IoT traffic.

The triggering of the end state in the state machine indicates that a bottleneck in the IoT network has been detected. A bottleneck in the context of the IoT network is any point in the application infrastructure which has breached the established QoS threshold. Since many applications operate on the network we can have multiple bottlenecks in the network.

C. Prediction Engine

We now discuss another important part of the Smart Testing Framework, the Prediction Engine, as shown in Fig. 3. The Prediction engine is used for predicting the resource utilization in a dynamic IoT network. The primary components of the engine are: the data processor, the predictor and the test case

generator. Together they provide users with the ability to build custom prediction algorithms and execute test cases to detect bottlenecks.

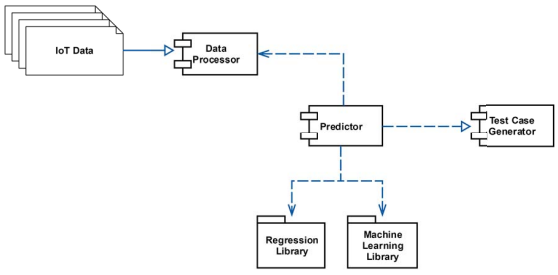

Fig. 3. Prediction Engine

1) ***Data Processor***
The input to the Data Processor is the historical resource usage information that has been collected previously from the IoT network. We can use this data to make predictions about usage patterns that have not yet been observed. The Data Processor ingests, validates, formats the data and passes it to the Predictor.
2) ***Predictor***
The role of the Predictor component is to apply a prediction algorithm on the incoming data that has been passed to it. For example, as shown in Fig. 3, we can import a regression library and use it to generate a predicted value (alternatively, a machine learning library can also be used for this purpose). In our experiments we used CPU utilization as the value we want to predict based on the number of active IoT devices on the network.
3) ***Test Case Generator***
The role of the Test Case Generator is to pass the action to the Smart Testing Framework. For example, this action can be a request to create a given number of IoT devices as specified by the Predictor.

III. IoT Devices

IoT devices come in a variety of types and existing legacy devices are being retrofitted with Internet connectivity. In this section we describe two common IoT device types and define the data structures that they emit.

A. IoT Temperature and Light Sensor

Environment monitoring is a common use case for IoT devices[5]. Therefore, to meet our research objectives of providing a platform for emulating IoT networks, we decided to implement an environmental IoT device. In our design we used the Texas Instruments (TI) SensorTag as the model for our virtualized IoT device that produces temperature and luminosity data[6]. The TI SensorTag has several embedded sensors that emit readings with various data types as shown in Table I. The data types are all numerical which means that from a size perspective, each set of emitted readings it is quite small.

Sensor Name	Unit	Type	Description
pressure	hPa	float	Pressure sensor air pressure
pressure_t	C	float	Pressure sensor temperature
humidity	%RH	float	Humidity sensor relative humidity
humidity_t	C	float	Humidity sensor temperature
objtemp	C	float	IR temp sensor object temperature
light	Lux	float	Light sensor illuminance
battery	mV	float	Battery voltage level

TABLE I
SensorTag embedded sensors

- *Features*
The device works by simply emitting readings for each sensor type at a given interval, for e.g. every 1 second. Since we know the data types and emission rates, we can reliably reproduce this data without the need for the physical device by designing a software version of the sensor.
- *Message Format*
To be able to process the data in a standardized way we have defined a message format that is based on the JSON standard. This allows us to validate the message integrity as it gets passed between applications on the IoT network. Fig.4 is a an example of a temperature reading. Since we use a common message format, it also allows us to store the information in the same database schema.

```
{
 sensorID:"IoT_temperature_sensor_65_3001"
 ,sensorType:"room_temperature"
 ,value:"16"
 ,timestamp:"1533663203"
 ,daydate:"20180807"
}
```

Fig. 4. Temperature Sensor Message Format

B. IoT Camera

Another type of IoT device that poses challenges for network designers are devices that generate large amount of traffic. An example of such a device is a camera. Cameras are an important tool in monitoring the environment and produce video streams that can be analyzed in real time. To understand the behavior of this type of IoT device, we created a prototype camera using the Raspberry Pi Kit that came with a camera module.

- *Features*
To create a virtualized representation of the IoT camera we documented several necessary characteristics for a fully working camera. The camera must produce video, must have a connection to the Internet and must be able to stream the video to a recipient. Towards this goal we created an application that stores a video file, implemented several libraries for image processing and

created a connector to a database to store the video data. The application creates streams by looping through the video file, breaking the video down into frames, encoding the images into a string based format and then writes this data to the database.

- *Message Format*
  The messaging format shown in Fig. 5 was created to transmit and store the data. We store each image in the database with a frame id. This is a concatenation of the camera id and timestamp. This allows us to reconstruct the video frame by frame if necessary. The advantage of storing the images by frame allows other applications to perform image recognition tasks without have to repeatedly break down the video into frames each time the camera data is queried.

```
{
   camera_id:"IoT_camera_68_3010"
  ,frame_id:"153467408132"
  ,value:"base64_encoded_value_here"
  ,timestamp:"1533663203"
  ,daydate:"20180807"
 }
```

Fig. 5. IoT Camera Data Format

IV. IoT Application

In this section we discuss the key modules that were created to support the major functionalities of the EMU-IoT application used in this work.

A. IoT Device Service

The IoT Device service provides the primary functionality for creating and destroying virtualized devices in EMU-IoT. The service abstracts many layers of the device creation process which eventually reaches the container service provider libraries. This function will carry out the actual tasks of creating the containers. This allows us in the future to change container service providers if necessary. In our example application we created IoT Temperature and Light Sensor and IoT Camera.

B. IoT Load Balancer

The IoT Load Balancer provides several functions for orchestrating the management of IoT Nodes and the assignment of IoT devices to these IoT Nodes. IoT Nodes are objects that represent physical/virtual machines which are the computing resources from the cloud service provider. The orchestration functions of IoT Load Balancer ensure that whenever a request to create a IoT device is made, the correct IoT Node is provided based on the Load Balancing policy. This is an important feature for EMU-IoT because we want to be able to execute experiments that are geographically distributed. For example, we can set a policy to force all newly created IoT Devices to a particular IoT Node in a certain geographic location. IoT Load Balancer is also responsible for maintaining the overall health of the IoT network. IoT Load Balancer can perform cleaning and reset functions in case an experiment did not execute completely and remove lingering IoT devices on the network. Lingering IoT Devices are emulated IoT devices that are no longer needed.

C. IoT Monitor

The IoT Monitor provides all of the data gathering capability from across the entire IoT Network. The gathered data can be statistics of the resources in use on a particular node, such as CPU, Memory, Disk and Network bandwidth utilization. We can also gather information about the container services such as the number of containers and whether they have terminated or are still running. In our implementation we collect all of the metrics mentioned above, but we only use the CPU utilization as the primary data source to make predictions. A major benefit of IoT Monitor is that the application was designed to be multi-threaded, therefore we can monitor many nodes and services independently and specific monitors can be disabled if not needed.

D. IoT Experiment

The IoT Experiment module is the main access point for the user to run coordinated experiments based on a set of configuration parameters that can be provided at build time. This module is configurable depending on the type of experiments that can be run. For example, in our case we ran experiments based on exhaustive search. This means that we target a particular CPU utilization value to find a bottleneck. In another case, we implemented a linear regression based experiment that learned from the exhaustive search data and then used the Smart Testing function to make predictions about unknown bottlenecks in an IoT application.

V. EXPERIMENTS

In this section we describe the types of experiments, the configuration details and how the data is collected. The experiments are divided into two phases. In the first phase as shown in Table II, we execute the exhaustive search up to a chosen CPU utilization target (represented by Target U in the table). This means that we obtain the CPU readings for all cases until we reach the target. In our experiment we have the Temperature and Light sensors which represents IoT devices that generate very small amounts of data. We repeat this experiment ten times for three utilization target levels.

Device	Runs	Type	Application	Target U
Temperature and Light	10	Exhaustive	Kafka	15.00%

TABLE II
EXPERIMENT PLAN: EXHAUSTIVE SEARCH

In phase two, we make predictions based on the data that is gathered in the first phase of the experiments. As shown in Table III, we chose prediction targets that are beyond the data collected so that we can predict unknown values. For example, for IoT temperature and light we collect data up to the 15% utilization target point and then we try to predict how many

Device	Runs	Type	Application	Target U
Temperature and Light	5	Regression	Kafka	17.50%
Temperature and Light	5	Regression	Kafka	20.00%
Temperature and Light	5	Regression	Kafka	22.50%

TABLE III
EXPERIMENT PLAN: LINEAR REGRESSION SEARCH

IoT devices are required to increase the CPU utilization up to 17.50%, 20.00%, and 22.50%. We repeat this process five times for three utilization target levels.

At the platform level, the software stack that is used across the network is Ubuntu 16.04 and the latest version of `Docker Community Edition 18.03.1-ce`[7]. At the application level we have `Apache Kafka 1.0.0`[8], `Spark 2.1.0`[9], and `Cassandra 3.1`[10] which are all deployed as containers on Docker. For the linear regression prediction function we used the `Python statsmodels` package[11].

VI. RESULTS

In this section we present the results of all the experiments that were executed on EMU-IoT. These experiments were intended to show that based on our research goals we could first measure the performance of an IoT application from end to end, and second, be able to perform bottleneck detection that would trigger an adaptation for a given IoT application. We discuss the experiment results for both the exhaustive search and linear regression cases.

A. IoT Temperature and Light Device

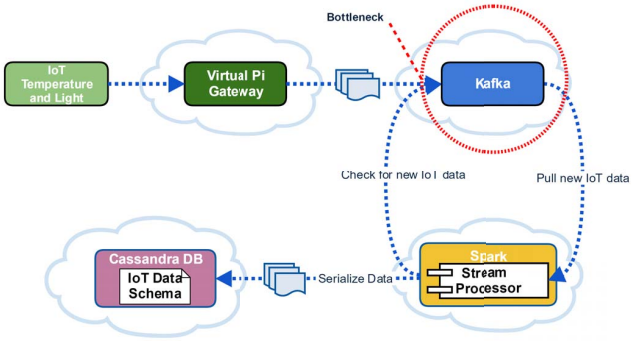

Fig. 6. Bottleneck Point - IoT Temperature and Light

In the IoT Temperature and Light experiment our goal is to find the bottleneck when running a lightweight type of IoT device. Once the bottleneck is found this would in theory trigger an adaptation in the IoT infrastructure (scale up, i.e. add more computing resources) represented in Fig. 6. For this type of IoT device the traffic is emitted and then sent to gateway where it is then forwarded to Kafka which is our aggregation point. From there the data is read by Spark and then stored in Cassandra. Prior to running these experiments we examined all three of these applications in the architecture and found that the CPU utilization increases the most with Kafka compared to the other applications. This is why we chose to examine only Kafka when experimenting with this IoT device type as shown in Fig. 6.

Exhaustive Search

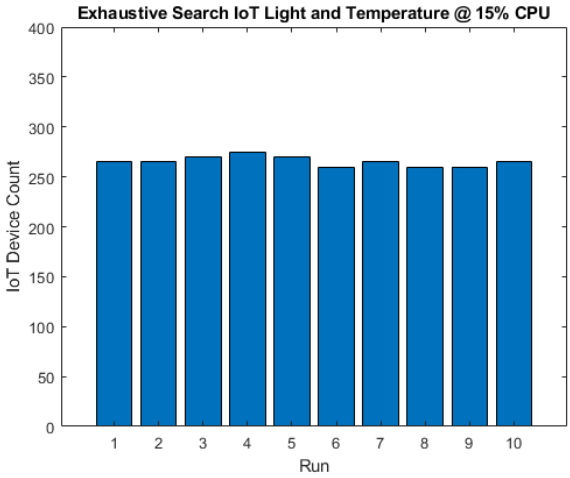

Fig. 7. Exhaustive Search Results IoT Temperature and Light

After executing the experiment 10 times, the results of the IoT Temperature and Light device type experiment show that there is little variation in the number of IoT devices required to hit the 15% CPU utilization target. This is shown in Fig. 7. The mean number of IoT devices over the 10 runs is 265 and the standard deviation is within approximately 2% of the mean. This suggests that there is little variance in the data. The most likely cause of the minor variation in the results is due to cloud variability and other background processes running inside the container.

Linear Regression

After executing the 10 runs from the exhaustive search we can derive a regression function from the data in an effort to predict values beyond the 15% CPU utilization target. We plot the data from the exhaustive search experiments shown in Fig. 8 and we can see that there is strong positive linear relationship between the IoT Temperature and Light device count and CPU utilization.

Using the regression function we execute three experiments by setting the target CPU utilization to 17.5%, 20%, and 22.5% and predicting the number of IoT Temperature and Light devices required to reach the chosen target utilization. Each experiment was run 5 times and the mean values obtained in the 5 runs are reported.

The prediction results obtained by linear regression are presented in Table IV. We first used the regression function to predict the number of IoT devices to reach the target CPU utilization level. Then we ran a validation experiment to

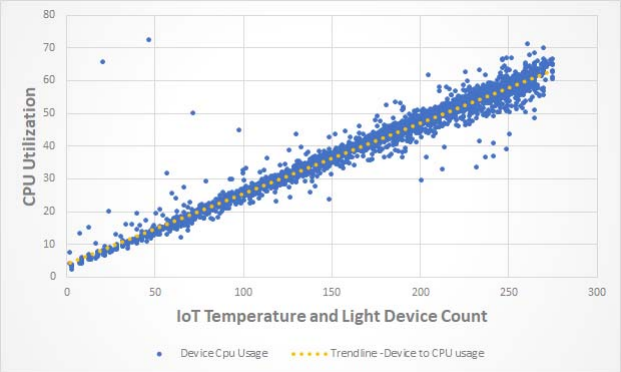

Fig. 8. Regression IoT Temperature and Light

Predicted number of Devices	Target U	Observed U	Std Dev
306	**17.50%**	**17.30%**	0.1328
352	**20.00%**	**19.85%**	0.3996
397	**22.50%**	**21.70%**	0.3884

TABLE IV
PREDICTION SUMMARY FOR IOT LIGHT AND TEMPERATURE

observe the CPU utilization at the given values of predicted IoT devices. The standard deviation across all 5 runs of the experiments for each target utilization level is also reported. As seen from Table IV, the target CPU utilization is very close to the observed CPU utilization for all predicted values of IoT devices. The standard deviation is also relatively low, suggesting low a variation in the results.

VII. RELATED WORK

Past researchers have proposed methods to improve QoS in IoT networks. Gotin et al. investigated different system utilization performance metrics as a scaling signal and compared them to messaging queues [12]. The authors found that monitoring the number of messages generated by IoT devices is a more precise predictor of resources needed. However, this requires instrumenting the application layer which makes the framework inflexible. For example, it will require modifying the application code to collect this information. In contrast, our Smart Testing Framework can work with any application without instrumenting it. Furthermore, in contrast to [12], our framework can test the system autonomously, i.e., it can generate test cases and monitor changes in the resources. Fehlmann et al. advocates for the need for real-time system testing in IoT networks, i.e., testing the behavior of a newly added IoT device in a live system [13]. The authors use a combinatory logic approach to determine how a test case should be generated.

Another approach to evaluating IoT networks has been to use simulation tools to test different scenarios. Ezdiani et al. proposed and implemented a testbed for IoT networks based on the concept of Quality of Service as a service (QoSaaS)[14]. Similar to our work they provide an environment to model heterogenous IoT networks. However the testbed is based on Contiki OS[15] and all simulations must run in that environment. In our approach we use EMU-IoT which uses Docker. This allows us to run our Smart Testing Framework on a wide variety of systems and hardware that support Docker.

VIII. CONCLUSION AND FUTURE WORK

Heterogeneous IoT networks are increasingly becoming more complex due to the high variability in IoT traffic caused by different IoT device types and time of use. A better understanding of the IoT traffic patterns can lead to improved QoS levels in IoT applications running on these networks. In this work, we implemented a Smart Testing Framework to detect bottlenecks and predict the resource utilization of a heterogeneous IoT network to maintain QoS in IoT applications. Our Smart Testing Framework can be used to predict resource usage as a trigger for infrastructure adaptation, i.e. adding or removing IoT devices from the network, in order to maintain desired QoS levels. Future work will include supporting new IoT device types such as wearable devices that generate movement data. This type of data can be emitted at high volumes but small size. We will also look at implementing machine learning algorithms that can continuously learn from past data to improve prediction accuracy.

REFERENCES

[1] A. Javed, K. Heljanko, A. Buda, and K. Frmling, "Cefiot: A fault-tolerant iot architecture for edge and cloud," in *IEEE WF-IoT*, 2018.
[2] H. F. Atlam, A. Alenezi, A. Alharthi, R. J. Walters, and G. B. Wills, "Integration of cloud computing with internet of things: Challenges and open issues," in *2017 IEEE International Conference on Internet of Things*, 2017.
[3] A. Botta, W. de Donato, V. Persico, and A. Pescap, "On the integration of cloud computing and internet of things," in *2014 International Conference on Future Internet of Things and Cloud*, 2014.
[4] G. White, A. Palade, C. Cabrera, and S. Clarke, "Quantitative evaluation of qos prediction in iot," in *2017 IEEE/IFIP DSN-W)*, 2017.
[5] B. Ramprasad, J. McArthur, M. Fokaefs, C. Barna, M. Damm, and M. Litoiu, "Leveraging existing sensor networks as iot devices for smart buildings," in *WF-IoT*, 2018.
[6] T. Instruments. (2018) Simplelink sensortag. [Online]. Available: http://www.ti.com/ww/en/wireless_connectivity/sensortag/
[7] Docker. (2018) Docker. [Online]. Available: https://www.docker.com/
[8] A. S. Foundation. (2018) Kafka a distributed streaming platform. [Online]. Available: http://kafka.apache.org/
[9] ——. (2018) Spark - lightning-fast unified analytics engine. [Online]. Available: https://spark.apache.org/
[10] ——. (2018) Cassandra. [Online]. Available: http://cassandra.apache.org/
[11] J. T. Josef Perktold, Skipper Seabold. (2018) Statsmodels for python. [Online]. Available: https://www.statsmodels.org/stable/index.html
[12] M. Gotin, F. Lösch, R. Heinrich, and R. Reussner, "Investigating performance metrics for scaling microservices in cloudiot-environments," ser. ACM/SPEC ICPE 2018.
[13] T. Fehlmann and E. Kranich, "Autonomous real-time software systems testing," in *Proceedings of the 27th International Workshop on Software Measurement and 12th International Conference on Software Process and Product Measurement*. ACM, 2017.
[14] S. Ezdiani, I. S. Acharyya, S. Sivakumar, and A. Al-Anbuky, "An architectural concept for sensor cloud qosaas testbed," in *Proceedings of the 6th ACM Workshop on Real World Wireless Sensor Networks*, ser. RealWSN '15. ACM, 2015.
[15] Contiki. (2018) Contiki: The open source os for the internet of things. [Online]. Available: http://www.contiki-os.org/