Supporting the Evaluation of Fog-based IoT Applications During the Design Phase

Jonathan Hasenburg
TU Berlin
Mobile Cloud Computing
hasenburg@tu-berlin.de

Sebastian Werner
TU Berlin
Information Systems Engineering
werner@tu-berlin.de

David Bermbach
TU Berlin
Mobile Cloud Computing
david.bermbach@tu-berlin.de

ABSTRACT

Fog application design is complex as it comprises not only the application architecture, but also the runtime infrastructure, and the deployment mapping from application modules to infrastructure machines. For each of these aspects, there is a variety of design options that all affect quality of service and cost of the resulting application. In this paper, we propose an approach for evaluating fog-based IoT applications already during an early design phase. Our approach relies on modeling and simulation to provide estimates for quality of service and cost so that developers can interactively choose a good application design.

CCS CONCEPTS

• **Software and its engineering → Integrated and visual development environments**; • **Computer systems organization** → *n-tier architectures*; Sensors and actuators;

KEYWORDS

Fog Computing, Application Design, Simulation, IoT

ACM Reference Format:
Jonathan Hasenburg, Sebastian Werner, and David Bermbach. 2018. Supporting the Evaluation of Fog-based IoT Applications During the Design Phase. In *5th Workshop on Middleware and Applications for the Internet of Things (M4IoT'18), December 10–11, 2018, Rennes, France.* ACM, New York, NY, USA, 6 pages. https://doi.org/10.1145/3286719.3286720

1 INTRODUCTION

The widespread deployment of connected devices in the Internet of Things (IoT) has substantially increased the amount of data available to developers. Today's IoT applications can make use of this data to enable more sophisticated application scenarios.

When designing an IoT application, the current go-to approach is collecting data at the edge, transmitting it to the cloud for processing, and sending the processed results back to the edge, e.g., to switch on a light in the presence of movement in a smart home scenario [15]. Due to its simplicity, this approach is used by many services, e.g., AWS IoT[1] or the Azure IoT Hub[2]. However, disadvantages include long response times, unnecessary data transmissions and the risk of exposing sensitive data to third parties [2].

Performing some tasks already at the edge, as done by AWS Greengrass[3], can reduce bandwidth consumption and enables the edge to keep operating in the presence of network partitions. However, this approach is limited by available processing capabilities as edge devices are often not powerful enough to run compute-intensive tasks.

An obvious solution to this problem is to leverage the compute power provided by stronger machines such as cloudlets [16, 17] within the core network [2]. This execution environment is commonly referred to as *fog* [2, 3] and consists of edge devices, machines within the core network, and the cloud.

When designing an application for the fog, developers typically have to consider the application architecture, the runtime infrastructure and the deployment mapping from application modules to infrastructure machines. For each of these three aspects, a number of design options exist and each option can be combined in various ways with options from the other aspects. This leads to a multitude of possible application design options.

Deciding on a particular design should be based on a careful evaluation of effects on quality of service (QoS) and cost. While such an evaluation tends to be complicated, we believe that it is worthwhile as the added benefits of efficiently using the fog can bring significant improvements to IoT applications. Within this paper, we aim to support such evaluations to help developers choose an design option. Therefore, we make the following contributions:

- We propose an approach for the evaluation of fog-based IoT applications that is applicable during the design phase.
- We present a proof of concept implementation of our approach called FogExplorer.
- We demonstrate how our approach can be used in an example scenario.

The remainder of this paper is structured as follows: In section 2, we introduce our approach. Then, we present a proof of concept implementation of our approach and demonstrate how it can be used in an example scenario (section 3). Finally, we discuss related work in section 4 before concluding in section 5.

2 APPROACH

When designing a fog-based IoT application, various options for the runtime infrastructure, application architecture, and possible deployment mappings lead to a multitude of possible designs. With

M4IoT'18, December 10–11, 2018, Rennes, France
© 2018 Association for Computing Machinery.
ACM ISBN 978-1-4503-6118-7/18/12…$15.00
https://doi.org/10.1145/3286719.3286720

[1]aws.amazon.com/iot/
[2]azure.microsoft.com/en-us/services/iot-hub/
[3]aws.amazon.com/greengrass/

our approach, we aim to provide developers with the means to evaluate these designs with the help of an iterative modeling and simulation process (section 2.1). Based on a simplified infrastructure model (section 2.2) and application model (section 2.3), we use simulations to interactively give QoS and cost estimates for changing deployment mappings (section 2.4). Such estimates allow developers to better understand implications of their design choices and balance QoS and cost.

2.1 Modeling and Simulation Process

Figure 1 shows the iterative modeling and simulation process: a developer first creates a high level infrastructure model (1a) and application model (1b). The infrastructure model describes machines and their interconnections; the application model defines application modules and all inter-module data streams. Note, that no implementation is required for the simulation process as application modules only contain very high level information on the later to be implemented software components of an application.

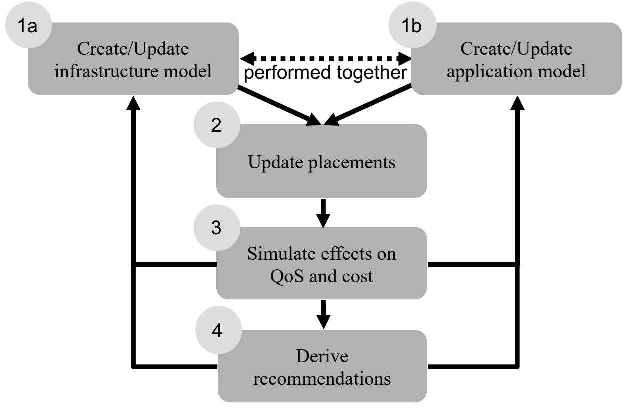
1a
Create/Update infrastructure model
1b
Create/Update application model
performed together
2
Update placements
3
Simulate effects on QoS and cost
4
Derive recommendations

Figure 1: The Iterative Modeling and Simulation Process

Based on a first infrastructure and application model, developers can then start to place application modules on infrastructure machines (2) to create a deployment mapping. Every placement update affects QoS as well as cost and should, hence, trigger a new simulation run (3). By studying these effects, recommendations on how to optimize placements and the infrastructure or application model can be derived (4). This information allows developers to iteratively improve their design and compare different design solutions.

2.2 Infrastructure Model

When designing a system, developers usually have only a vague idea about their runtime infrastructure. Therefore, we cannot assume to have detailed information or benchmarking results on available performance. Thus, the model has to support a high level, abstract description of infrastructure which is accomplished by focusing on six properties: three for machines, and three for connections between machines.

For every machine, the property *performanceIndicator* is a rough estimate on the performance compared to a reference machine[4], i.e., a performance indicator of 4.0 means that the respective machine is about four times "faster" than the reference machine. Besides the performance indicator, machines have properties that describe the amount (*availableMemory*) and price (*memoryPrice*) of available memory. If necessary, our model can be extended with additional properties, e.g., to describe storage capacities and prices or QoS properties such as the availability. However, we explicitly decided to use a simple model as our approach targets the design phase.

For every connection, the *latency* property gives an estimate on the latency between two machines. Furthermore, *availableBandwidth* and *bandwidthPrice* specify amount and price of available bandwidth. A machine may have multiple connections to another machine so that incoming and outgoing data can be modeled separately and slow but inexpensive as well as fast but expensive connections can be considered.

2.3 Application Model

IoT applications are often event-driven: "things" produce data streams, the data streams are analyzed to identify certain events, and events trigger actions on other "things". In the fog, each of these functions is potentially executed at a different location. Therefore, an application architecture should build on self-contained modules which can be run individually and exchange information by interconnecting data streams, as also done in [5, 7, 18].

For such application modules, we identified three distinct types: *source*, *service* and *sink*. Sources produce data, services process data and forward results, and sinks receive data. Figure 2 shows an illustrative application model with four distinct application modules: one source, one service, and two sinks. As our approach aims to enable model definitions very early during the design process, we again decided to use only a limited set of module properties.

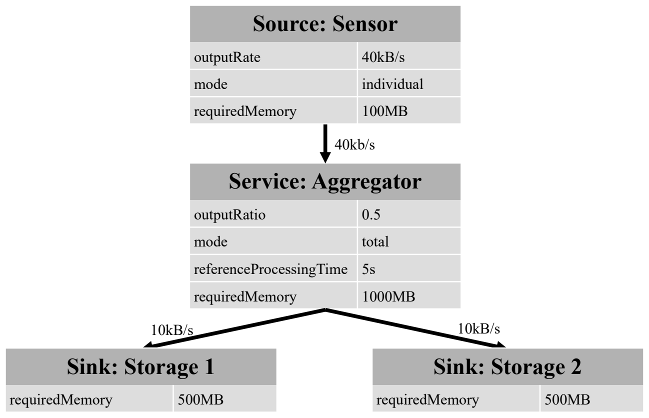

Figure 2: Example of an Application Model

In our model, sources, e.g., IoT sensors, produce a constant and continuous stream of data at a defined rate (*outputRate*) which is processed by a sequence of services in a certain amount of time (*referenceProcessingTime*). With this abstraction, developers only have to specify the amount of data produced per time and do not need to be concerned about details such as data transmission intervals or varying data stream volumes. E.g., in figure 2 the Sensor source

[4]A developer can chose any machine of the runtime infrastructure to be the reference machine, e.g., a Raspberry Pi or a specific AWS EC2 instance type.

sends 40kB/s of temperature sensor readings to the Aggregator service which needs five seconds to aggregate the data before its results are forwarded to the two sinks Storage 1 and Storage 2. The service output rate is dynamically calculated by multiplying the amount of incoming data with the respective *outputRatio* which defines how the service changes the incoming data stream volume. In the example, the Aggregator service has an output rate of 20kB/s ($40kB/s * 0.5$).

Note, that services do not "block" while processing data, i.e., they continuously receive and process data. As the actual processing time can only be determined by benchmarking real application modules on physical infrastructure – which is not feasible during the design phase – we simplify this problem by only requiring developers to estimate how long the processing would take on a reference machine. With this information we can estimate the actual processing time based on the performance indicator of the machine the module is placed on and the reference processing time of the module. For example, the Aggregator of figure 2 is estimated to require 5s for the processing of data on the reference machine which has a performance indicator of 1.0.

Every service and source has a *mode* which describes how the application module output rate is distributed across outgoing data streams. Either every individual subsequent module receives the full calculated output rate (mode = "individual"), or the total amount of data produced by the module is uniformly split across all outgoing data streams (mode = "total"). For example, if the Aggregator service from figure 2 had four outgoing data streams instead of two, each data stream would contain 5kB/s as the calculated output rate of 20kB/s would need to be divided by four. Lastly, each module has the property *requiredMemory* which defines how much memory it will use at runtime.

Developers do not have to specify any properties for data streams, except which modules they connect. The amount of required bandwidth (*requiredBandwidth*) can be derived from output rates, output ratios and specified module modes.

2.4 Simulation

Based on the infrastructure and application model, developers should be able to interactively evaluate the effects of a module placement option on QoS and cost. With the current model properties, we simulate effects on four metrics: *processing cost*, *processing time*, *transmission cost*, and *transmission time*. The cost metrics describe the average cost per second in a given setup; the time metrics describe how long it takes to process a single data item on a certain machine or to transmit it over a connection. We propose to use a tool implementation to determine these metrics. Such an interactive simulation tool has to run a number of calculations after each module placement, which we describe below. As an illustrating example, we continue to use the application model introduced in figure 2.

1. Data stream routing: The tool has to determine which connections are used by each of the modules' data streams, i.e., are involved in the information exchange with other modules. E.g, let us consider an infrastructure model with machines A, B, and C; A is connected to B and B is connected to C. When the Sensor service is placed on A and the Aggregator service is placed on C, the data stream between those two modules involves the A-B connection and the B-C connection. In more complicated cases with multiple possible connection paths, the tool can use a shortest path algorithm such as Dijkstra [8] to determine the set of connections with the cheapest bandwidth price.

2. Resource usage: The tool has to calculate how a placement affects bandwidth and memory utilization of machines and connections. The new amount of used memory equals the old amount plus the amount required by the placed module. Similarly, for each of the module's data streams, the new amount of used bandwidth for each connection equals the connection's old amount plus the amount required by the data stream. If a module is removed from a machine, bandwidth and memory utilization are reduced by the respective values. For instance, if machine C uses 500MB of memory, the additional placement of the Aggregator service on C results in a total usage of 1500MB, and connections A-B and B-C both use a bandwidth of 40kB/s.

3. Under-provisioning: The tool has to calculate the under-provisioning ratio (*underProvisionRatio*) for each connection and each machine, which describes the proportion between required and available resources. If more resources are available than used, the ratio equals 1. Otherwise, it equals the amount of used resources (bandwidth or memory) divided by the amount of available resources, e.g., the placement of the Aggregator service on a machine with 100MB of memory would lead to an under-provisioning ratio of 10 ($1000MB/100MB = 10$).

4. Individual cost metrics: The tool has to calculate transmission cost and processing cost for each data stream and application module. For this, it has to consider under-provisioning of resources. The transmission cost equals $\frac{requiredBandwidth}{underProvisionRatio} * bandwidthPrice$ and the processing cost equals $\frac{requiredMemory}{underProvisionRatio} * memoryPrice$.

5. Individual time metrics: The tool has to calculate transmission time and processing time for each data stream and module. If the respective resource is under-provisioned, each time metric is infinite because the system would experience a backpressure that can never be handled as the input always exceed the transmission or processing capabilities. Otherwise, the transmission time per connection equals its latency, and the processing time equals the machine's estimated processing time (which considers the machine's performance indicator).

6. Total metrics: The tool has to calculate a total metric for each individual cost and time metric with which developers get a first impression on a design's characteristics by studying a single value. E.g., the total processing cost equals the sum of each module's processing cost.

7. Recommendations: The tool has to provide developers with recommendations on how to optimize placements and the current infrastructure or application model. These recommendations can be based on information such as under-provisioned machines and connections. Additionally, the recommendations have to help identify invalid module placements where a missing infrastructure connection disrupts data streams. Furthermore, recommendations have to highlight which machine and connection resources could be reduced without affecting QoS or cost.

We opted for this set of metrics, as they give a good overview about effects on QoS and cost for a particular design. In addition, our tool can calculate these metrics before a developer has full information about the runtime infrastructure and before the application is implemented, as we only require some modeling data about the application components and the planned runtime infrastructure. For the price of an increased complexity, it is also possible to extend the infrastructure and application model properties if more extensive QoS and cost metrics are required, or additional recommendations should be derived.

3 EVALUATION

The evaluation of our proposed approach is twofold. We first present our proof of concept implementation (section 3.1). Then, we demonstrate how our approach can be used in an example scenario (section 3.2).

3.1 Proof of Concept Implementation

As a proof of concept, we implemented FogExplorer, an interactive simulation tool that follows our approach. FogExplorer is available as open source[5] and is built as a front-end javascript application that utilizes ECMAScript 2015 features. The tool runs without a backend or web server, so only a modern web browser is required for the execution of FogExplorer. For the graph visualizations, we used the network visualization capabilities of vis.js[6], the front end is built with Foundation[7] and jQuery[8]. We also implemented a node.js[9] package that can be used for automatic model evaluations without the visual front end.

3.2 Scenario

In this section, we apply our modeling and simulation process from section 2.1 to an example scenario. Due to page limitations, we decided to use an edge computing scenario as an example. The overall process, however, is similar for more advanced scenarios in which machines are geo-distributed across edge, core network and cloud.

In our planned scenario, a company wishes to control the climate in one of their buildings based on sensor readings to reduce energy cost. Multiple possible designs for such a *smart building* application exist, so the company needs to understand effects on QoS and cost to decide which design should be implemented.

For the climate in a building, many factors are important and thus should be monitored, e.g., wind force, direct sunlight exposure, inside and outside temperature, or air pollution. Furthermore, many climate control mechanisms exist, e.g., opening and closing windows, shades, and shutters, air conditioning, or heating. In addition, weather forecasts and historic data enable more intelligent decisions [13]. For instance, when a cold front is approaching, it makes sense to wait a few more minutes rather than cooling down the temperature with the help of air conditioning if temperature thresholds have only been exceeded slightly.

For our evaluation, we focus only on the cooling aspect of the scenario. Moreover, we only monitor the inside and outside temperature and employ two climate control mechanisms: airing and air conditioning. In other words, when possible the building should be cooled down by opening windows at appropriate times, i.e., when the outside temperature is lower than inside. The energy-consuming air conditioning should only be used as a last resort.

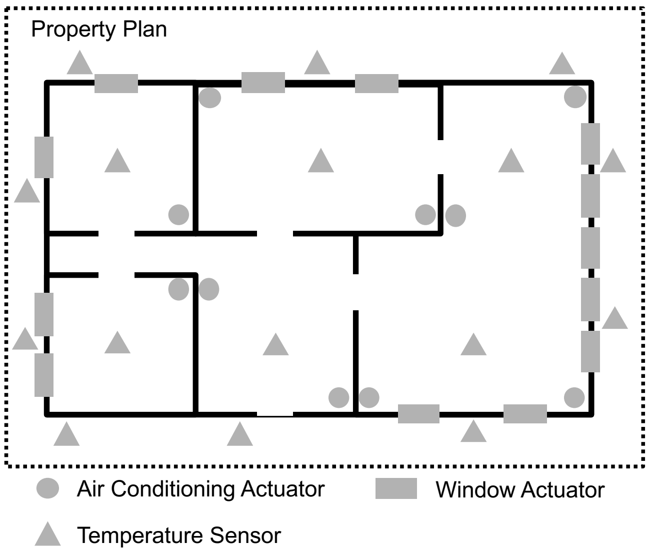

Figure 3: General Scenario Setup

The general scenario setup is shown in figure 3. A number of temperature sensors located inside and outside of the building emit measurements every second. Each window of the building is equipped with an actuator that can open and close the window. Furthermore, a number of air conditioning units are distributed across the building which are also controlled by actuators.

The main concern of our scenario evaluation is selecting a suitable runtime infrastructure for the IoT application controlling the actuators. We will evaluate two possible runtime infrastructures: decentralized and centralized. In the decentralized setup, edge devices exchange information directly, whereas they only communicate with a (central) server that also does most of the processing in the centralized setup.

Each runtime infrastructure option comes with its own set of advantages and disadvantages. For example, decentralized setups naturally have no single point of failure and can usually cope well with network partitioning. In the centralized setup, data access and debugging become easier. None of these options is superior in every situation, so a case by case evaluation is required. The approach proposed in this paper aims to enable this evaluation and

[5]https://github.com/OpenFogStack/FogExplorer
[6]https://visjs.org
[7]https://foundation.zurb.com
[8]https://jquery.com
[9]https://nodejs.org/en/about/

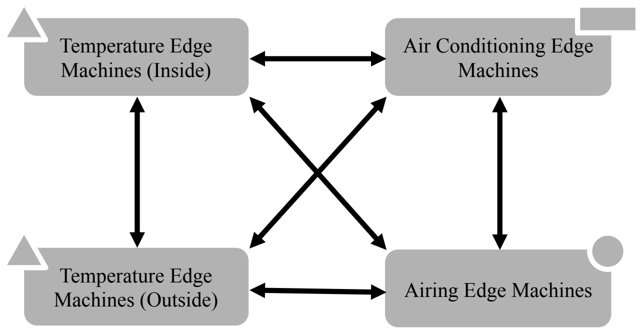
Temperature Edge Machines (Inside)
Air Conditioning Edge Machines
Temperature Edge Machines (Outside)
Airing Edge Machines

Figure 4: The Decentralized Infrastructure Model

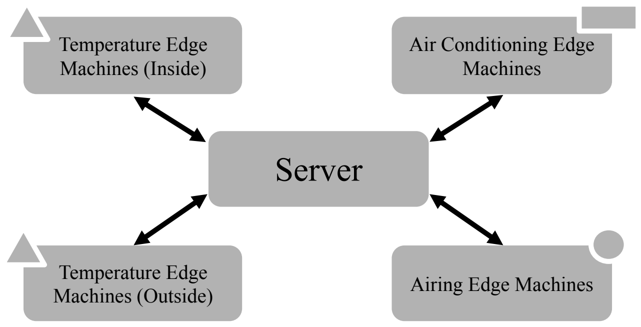
Temperature Edge Machines (Inside)
Air Conditioning Edge Machines
Server
Temperature Edge Machines (Outside)
Airing Edge Machines

Figure 5: The Centralized Infrastructure Model

supports decisions with limited information on application architecture and runtime infrastructure as the evaluation has to occur early during the design phase. Furthermore, our approach remains flexible enough to evaluate the impact of various adjustments such as scaling the number of edge devices or changing resource price estimates.

Infrastructure and Application Models

The first step in our approach is to create an initial infrastructure model for each runtime infrastructure. To reduce the model's complexity, we aggregated all edge machines connected to the same type of "thing" into a single machine. While this is only possible when all machines and their connections have comparable hardware resources, we could still create an individual machine for each edge device, if different resources were to be used.

Figure 4 shows our decentralized infrastructure model. Here, powerful edge machines are directly connected to the building's sensors and actuators and can communicate with all other edge machines directly. Furthermore, edge machines connected to sensors analyze measurements and emit events, e.g., when a value is lower than a threshold or unusual high. This preserves bandwidth. Edge devices connected to actuators analyze received events and instruct the local actuator based on their analysis. However, each edge machine requires processing and storage capabilities.

Figure 5 shows our centralized infrastructure model. Here, the building's sensors are connected to weak edge devices that send each measurement to a central server for processing. The server analyzes measurements and sends instructions to the edge devices which in turn only forward them to actuators without additional processing. While these weak edge devices do not require extensive processing and storage capabilities, the central server does. In addition, as all raw sensor data needs to be transmitted to the central server, the bandwidth utilization is relatively high compared to the decentralized setup.

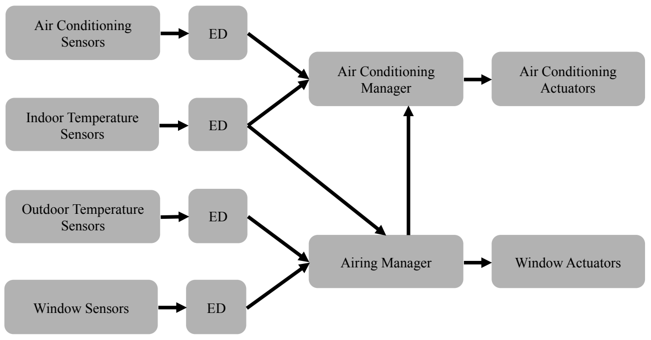

Figure 6: The Application Model

Besides the infrastructure model, our approach requires an application model. For the scenario, we created the one presented in figure 6. It is compatible with both runtime infrastructures. Sensor sources continuously send data to event dispatcher (ED) services which check whether a new event has to be dispatched. Both the Air Conditioning Manager and Airing Manager services receive events and analyze them. Based on the analysis, they send instructions to the two actuator sinks. Note that every actuator "thing" has a matching sensor as the managers needs to read the current state, e.g., whether an air conditioning unit is currently active.

Simulation

Having created both models, developers can begin to place sources, services and sinks on machines. For both runtime infrastructures, sources and sinks are placed on edge devices. In the decentralized setup, ED and Manager services are also placed on the edge, while all services are placed on the server in the centralized setup. With these deployment mappings, developers of the smart building can begin to study how changing properties of the application and infrastructure models affect the QoS and cost metrics.

In the end, which runtime infrastructure is "better" depends on the many factors of a concrete scenario. These include the number of sensors and actuators, frequency and volume of sensor measurements, the tasks and analysis run by EDs and Manager services, as well as cost for processing and transmission resources. By following our approach, developers can run the necessary simulations in order to receive recommendations, iteratively improve their design, and compare different design solutions.

As the evaluation presented in this paper is done using a case study, we prepared a demo[10] for this scenario utilizing our proof of concept implementation FogExplorer. In this demo, readers can

[10]https://openfogstack.github.io/FogExplorer/

evaluate effects on QoS and cost for the centralized and decentralized runtime infrastructure and experiment with different model properties. Table 1 shows the results for the default values in the demo. In this particular case, the centralized infrastructure has higher cost but better QoS metric values, so if such a quality level is required, the centralized runtime infrastructure should be chosen. In conclusion, the data provided by FogExplorer allows developers to make informed decisions on matters such as the choice for a particular runtime infrastructure without the need for prototype implementations early on.

Table 1: Results from the Demo Evaluation

	Centralized	Decentralized
Total transmission cost	3.33 cent/min	0.28 cent/min
Total transmission time	6 sec	4 sec
Total processing cost	0.04 cent/min	0.01 cent/min
Total processing time	0.85 sec	3.4 sec

4 RELATED WORK

Fog computing is still a new field with many open issues and research questions [2, 9, 14]. While many efforts have been made to define what fog computing is and what it is good for, the number of publications that propose solutions for the evaluation of fog application designs is still limited.

With iFogSim [12], Gupta et al. present a modeling and simulation tool for Fog environments which aims to simulate the effects of different resource management algorithms such as the ones presented in [1, 10, 11]. However, iFogSim dose not evaluate application and infrastructure designs. Furthermore, it requires access to excessive monitoring data which is not available during an early design phase. Additionally, it only supports strictly hierarchical runtime infrastructures, as their tool is based on Cloudsim [6].

Brambilla et al. [4] propose a methodology for simulating mobility, networking, and energy IoT devices in urban environments. As it aims to help understand how IoT systems with a large number of interconnected devices behave, their methodology is not suited for modeling and evaluation of application architecture or runtime infrastructure designs.

Cardellini et al. [7] propose an approach for the optimal operator placement of distributed applications based on user-specific QoS metrics. While we use a comparable application and infrastructure model, their approach cannot be used in an early design phase where only limited data is available.

Similarily, Brogi and Forti [5] propose an approach that also uses a comparable application and infrastructure model to automatically determine eligible deployment mappings. However, their approach does not consider cost metrics and cannot be used to interactively compare different design and mapping options.

5 CONCLUSION

In this paper, we proposed an approach for the evaluation of fog-based IoT applications that can be used already during the design phase. When following our approach, developers can compare different design options based on QoS and cost metrics. This approach builds upon a modeling and simulation process in which runtime infrastructure and application architecture models are created, and the effects of application module placements on infrastructure machines are simulated. As only limited information might be available for the modeling, our proposed simulation procedure only requires high level model definitions and can therefore be used early in the development process. As a proof of concept, we presented FogExplorer, our prototypical implementation, and demonstrated how our approach can be used in an example scenario.

ACKNOWLEDGMENTS

This work partly been supported by the European Commission through the Horizon 2020 Research and Innovation program under contract 731945 (DITAS project).

REFERENCES

[1] Hamid Reza Arkian, Abolfazl Diyanat, and Atefe Pourkhalili. 2017. MIST: Fog-based Data Analytics Scheme with Cost-efficient Resource Provisioning for IoT Crowdsensing Applications. 82 (2017), 152–165.

[2] David Bermbach, Frank Pallas, David García Pérez, Pierluigi Plebani, Maya Anderson, Ronen Kat, and Stefan Tai. 2018. A Research Perspective on Fog Computing. In *2nd Workshop on IoT Systems Provisioning & Management for Context-Aware Smart Cities (ISYCC)*, Vol. 10797. Springer, 198–210.

[3] Flavio Bonomi, Rodolfo Milito, Jiang Zhu, and Sateesh Addepalli. 2012. Fog Computing and its Role in the Internet of Things. In *Proc. of the First Edition of the MCC Workshop on Mobile Cloud Computing - MCC '12*. ACM Press, 13.

[4] Giacomo Brambilla, Marco Picone, Simone Cirani, Michele Amoretti, and Francesco Zanichelli. 2014. A Simulation Platform for Large-Scale Internet of Things Scenarios in Urban Environments. In *Proceedings of the The First International Conference on IoT in Urban Space*. ICST.

[5] Antonio Brogi and Stefano Forti. 2017. QoS-Aware Deployment of IoT Applications Through the Fog. 4, 5 (2017), 1185–1192.

[6] Rodrigo N. Calheiros, Rajiv Ranjan, Anton Beloglazov, César A. F. De Rose, and Rajkumar Buyya. 2011. CloudSim: A Toolkit for Modeling and Simulation of Cloud Computing Environments and Evaluation of Resource Provisioning Algorithms. 41, 1 (2011), 23–50.

[7] Valeria Cardellini, Vincenzo Grassi, Francesco Lo Presti, and Matteo Nardelli. 2016. Optimal Operator Placement for Distributed Stream Processing Applications. In *Proceedings of the 10th ACM International Conference on Distributed and Event-based Systems - DEBS '16*. ACM Press, 69–80.

[8] E. W. Dijkstra. 1959. A Note on Two Problems in Connexion with Graphs. 1, 1 (1959), 269–271.

[9] Manuel Díaz, Cristian Martín, and Bartolomé Rubio. 2016. State-of-the-art, Challenges, and Open Issues in the Integration of Internet of Things and Cloud Computing. 67 (2016), 99–117.

[10] Lazaros Gkatzikis and Iordanis Koutsopoulos. 2013. Migrate or Not? Exploiting Dynamic Task Migration in Mobile Cloud Computing Systems. 20, 3 (2013), 24.

[11] Lin Gu, Deze Zeng, Song Guo, Ahmed Barnawi, and Yong Xiang. 2017. Cost Efficient Resource Management in Fog Computing Supported Medical Cyber-Physical System. 5, 1 (2017), 108–119.

[12] Harshit Gupta, Amir Vahid Dastjerdi, Soumya K. Ghosh, and Rajkumar Buyya. 2017. iFogSim: A Toolkit for Modeling and Aimulation of Resource Management Techniques in the Internet of Things, Edge and Fog Computing Environments. 47, 9 (2017), 1275–1296.

[13] Benjamin Karg and Sergio Lucia. 2018. Deep Learning-Based Embedded Mixed-Integer Model Predictive Control. (2018), 2075–2080.

[14] Carla Mouradian, Diala Naboulsi, Sami Yangui, Roch H. Glitho, Monique J. Morrow, and Paul A. Polakos. 2018. A Comprehensive Survey on Fog Computing: State-of-the-Art and Research Challenges. 20, 1 (2018), 416–464.

[15] Lukas Reinfurt, Uwe Breitenbücher, Michael Falkenthal, Frank Leymann, and Andreas Riegg. 2016. Internet of Things Patterns. In *Proc. of the 21st European Conference on Pattern Languages of Programs - EuroPlop '16*. ACM Press, 1–21.

[16] Mahadev Satyanarayanan, Paramvir Bahl, Ramon Caceres, and Nigel Davies. 2009. The Case for VM-based Cloudlets in Mobile Computing. (2009), 9.

[17] Mahadev Satyanarayanan, Grace Lewis, Edwin Morris, Soumya Simanta, Jeff Boleng, and Kiryong Ha. 2013. The Role of Cloudlets in Hostile Environments. 12, 4 (2013), 40–49.

[18] Anshu Shukla, Shilpa Chaturvedi, and Yogesh Simmhan. 2017. RIoTBench: A Real-time IoT Benchmark for Distributed Stream Processing Platforms. 29, 21 (2017), e4257. arXiv:1701.08530