Poster: An Assessment Framework for Edge Applications

Martin Wagner, Julien Gedeon, Karolis Skaisgiris, Florian Brandherm, Max Mühlhäuser
Telecooperation Lab, Technische Universität Darmstadt, Germany
Email: {wagner, gedeon, skaisgiris, brandherm, max}@tk.tu-darmstadt.de

***Abstract*—We introduce an assessment framework for edge computing applications. The framework allows developers to measure the execution time of their applications in different environments and generate a model for the prediction of execution times. Based on these measurements and predictions, better informed management decisions can be made for edge applications.**

I. Introduction

Mobile Edge Computing (MEC) [1], [2] and emerging 5G networks allow users the usage of new mobile applications and services with low latencies and high bandwidth. To achieve this, computation resources in close proximity to users are made available. Edge nodes in such an edge infrastructure are typically equipped with heterogeneous hardware. In comparison to cloud datacenters, such edge nodes are heavily limited in their available computing resources. Because of these characteristics, they also lack the ability to easily scale to changing computing demands from users. In order to mitigate these problems, edge applications can be migrated between edge nodes. Furthermore, edge applications can offer multiple versions of themselves with different performance requirements [3]. Management decisions, such as where to migrate an edge application to or which application version to use, require a good estimate of application performance, especially a good estimate of execution time.

In this paper, we present an assessment framework for edge applications. It allows application developers to measure their applications so that optimized management decisions can be performed based on the measured application performance. Furthermore, we study the accuracy of performance assessments that are derived from previous measurements. We derive these assessments via linear regression and can use it to acquire a multitude of new application assessments without the need of additional measurements.

II. The Assessment Framework Design and Implementation

The assessment framework consists of (1) an execution time measurement tool and (2) a model generator. In order to obtain models with which a developer can automatically assess their edge applications, they first need to provide the assessment framework with some initial measurements of their applications. To do so, the developer uses the measurement tool provided by the assessment framework. They have to define different environments in terms of hardware resources and network conditions under which the application is then executed. The measurement tool then measures the time required for the execution of the application. These measurements will be fed into the model generator to generate a model with which the execution time of the application can be predicted for unknown, not before seen requirements. Based on these predictions, management decisions regarding migration and version switching can be made to react to changes in the execution and network environment. We expect the developer to perform only a small number of initial measurements since this step is an additional burden in the development process. Even though the accuracy of the model increases with additional measurements, we think that, in order to raise acceptance of developers for our framework, the burden on them should be minimized.

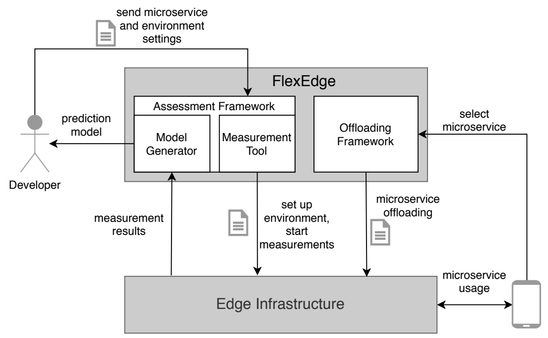


Figure 1. Overview of the framework architecture

We implemented a prototype of our assessment framework as an extension to *FlexEdge*, a container-based microservice offloading framework for edge computing [4]. It allows users to select one or several microservices from a microservice repository that are then offloaded to and executed on an edge infrastructure that is part of the system. Figure 1 illustrates an overview of the framework architecture. We make use of the Docker features of limiting the available memory and the cpu resources a container can use to be able to provide applications with different execution environments. In particular, we use the Docker runtime options *memory, cpu-period,* and *cpu-quota* [5] to constrain the available hardware resources of an application. This way, the execution time of a microservice can be measured under different hardware setups by limiting the resources of its underlying container. Furthermore, this lets us simulate multi-tenant deployments where each application is allocated a fraction of the available hardware resources. For each microservice, a message queue is created on which the service receives its input and another queue to which the service sends its output from where it can then be received by the user. We define the execution time of a microservice as the time from the receiving of its input to the sending of its output. Accordingly, this time is measured in our prototype. After the initial measurements have been performed, they will be sent to the model generator of our assessment framework. In our initial prototype, we use the ordinary least squares linear regression from *scikit learn* [6] to generate a model.

III. Experiments and Results

For our experiments, we measured the execution time of a face detection microservice for a multitude of different hardware resource constraints. In each of the measurements, we set the parameter *cpu-period* to 100 000 while setting the parameters *cpu-quota* to values between 10 000 and 200 000 and *memory* to values between 8 MB and 16 384 MB. The measurements were conducted on a Lenovo ThinkCentre M920X Tiny with an Intel Core i7-8700 and 16GB RAM running Ubuntu 18.04. To simulate a developer only performing a small number of initial measurements, we used 20 of these measurements to train a model using linear regression. In particular, we used the measurements for memory values of 8, 32, 128, 2 048, and 16 384 MB and for cpu-quota values of 10 000, 20 000, 30 000, and 50 000. Figure 2 shows the measured execution times and the, via linear regression, predicted execution times. Additionally, Figure 3 shows the prediction error of the trained model for the 20 values that were used to train this model as a heatmap. The prediction error is calculated as the absolute difference between the measured and predicted execution time. For the highest error, with a cpu-quota of 10 000 and memory of 8 MB, the measured execution time is 13 257 ms and the predicted time is 7 831 ms. For the lowest error, with a cpu-quota of 30 000 and memory of 8 MB, the measured time is 4 187 ms and the predicted time is 4 009 ms. A model trained with 20 initial measurements can result in execution time estimations which help in making better informed management decisions for edge applications. Our measurements have shown that increasing the cpu-quota past 50 000 does not yield a further decrease in execution time. But since our model is based on linear regression, it predicts for these cpu-quota values decreasing execution times, to the point of negative execution time predictions. This reveals a weakness of only performing well on a subset of all the possible hardware resource constraints.

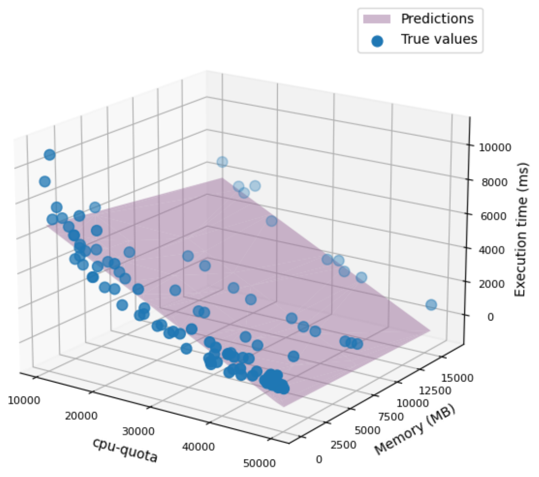
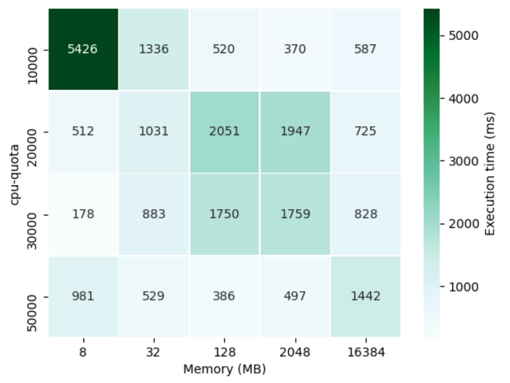


Figure 2. Predicted and measured execution times for different Docker runtime options



cpu-quota \ Memory (MB)	8	32	128	2048	16384
10000	5426	1336	520	370	587
20000	512	1031	2051	1947	725
30000	178	883	1750	1759	828
50000	981	529	386	497	1442

Figure 3. Prediction error for different Docker runtime options

IV. Outlook

In the previous section, we have shown our initial results with our assessment framework and a simple model based on linear regression. For our future work, we plan to extend our prototype with additional methods to generate an execution time prediction model. In particular, we want to include polynomial regression and nonlinear regression and investigate their performance. Furthermore, we want to investigate the impact of the number of initial measurements we expect the developer to perform. This is a trade-off between the burden on the developer and the accuracy of the resulting model. To better estimate the execution time of edge applications, we want to extend our framework with additional features for the model generation such as the availability of a GPU and the size of the applications input. Both of these features influence the execution time. Our prototype's current predictions are only valid for the hardware platform the measurements were performed on. We want to investigate the possibility of predicting the execution time for different hardware platforms. In summary, this work shows the potential of our initial prototype and with our planned work, we expect further improvements to our framework.

Acknowledgment

This work has been cofunded by the German Research Foundation (DFG) and the National Nature Science Foundation of China (NSFC) joint project under Grant No. 392046569 (DFG) and No. 61761136014 (NSFC), and as part of the CRC 1053 MAKI (DFG).

References

[1] M. Satyanarayanan, “The Emergence of Edge Computing,” *IEEE Computer*, vol. 50, no. 1, pp. 30–39, 2017.

[2] J. Gedeon, F. Brandherm, R. Egert, T. Grube, and M. Mühlhäuser, “What the Fog? Edge Computing Revisited: Promises, Applications and Future Challenges,” *IEEE Access*, vol. 7, pp. 152 847–152 878, 2019.

[3] S. Gholami, A. Goli, C.-P. Bezemer, and H. Khazaei, “A Framework for Satisfying the Performance Requirements of Containerized Software Systems Through Multi-Versioning,” in *Proc. of the International Conference on Performance Engineering (ICPE)*, 2019, pp. 1–11.

[4] J. Gedeon, M. Wagner, J. Heuschkel, L. Wang, and M. Mühlhäuser, “A Microservice Store for Efficient Edge Offloading,” in *Proc. of the IEEE Global Communications Conference (GLOBECOM)*, 2019, pp. 1–6.

[5] “Runtime options with Memory, CPUs, and GPUs,” accessed: 2020-08-26. [Online]. Available: https://docs.docker.com/config/containers/resource_constraints/

[6] “scikit learn Linear Regression,” accessed: 2020-08-26. [Online]. Available: https://scikit-learn.org/stable/modules/generated/sklearn.linear_model.LinearRegression.html