

Gathering Bicycle Ride Data and Using it for the Good

vorgelegt von

M.Sc.

Ahmet-Serdar Karakaya

ORCID: 0000-0002-2040-2782

an der Fakultät IV – Elektrotechnik und Informatik

der Technischen Universität Berlin

zur Erlangung des akademischen Grades

Fakultät IV

Technische Universität Berlin

Promotionsausschuss:

Vorsitzender: Prof. Dr. Stefan Tai

Gutachter: Prof. Dr. David Bermbach

Gutachter: Prof. Dr. Heather Kaths

Gutachter: Prof. Dr. Odej Kao

Tag der Wissenschaftlicher Aussprache: 21.03.2024

März 2024

Sworn Affidavit

I hereby declare that the thesis submitted is my own, unaided work, completed without any unpermitted external help. Only the sources and resources listed were used.

The independent and unaided completion of the thesis is affirmed by affidavit:

Berlin, January 1st, 1970

Jana Eggers

Abstract

FaaS is a cutting-edge new service model that has developed with the current advancement of cloud computing. It allows software developers to deploy their applications quickly and with needed flexibility and scalability, while keeping infrastructure maintenance requirements very low. These benefits are very desirable in edge computing, where ever changing technologies and requirements need to be implemented rapidly and the fluctuation and heterogeneity of service consumers is a considerable factor. However, edge nodes can often provide only a fraction of the performance of cloud computing infrastructure, which makes running traditional FaaS platforms infeasible. In this thesis, we present a new approach to FaaS that is designed from the ground up with edge computing and IoT requirements in mind. To keep it as lightweight as possible, we use CoAP for communication and Docker to allow for isolation between tenants while re-using containers to achieve the best performance. We also present a proof-of-concept implementation of our design, which we have benchmarked using a custom benchmarking tool and we compare our results with benchmarks of Lean OpenWhisk, another FaaS platform for the edge. We find that our platform can outperform Lean OpenWhisk in terms of latency and throughput in all tests but that Lean OpenWhisk has higher success rates for a low number of simultaneous clients.

Zusammenfassung

FaaS ist ein innovatives neues Servicemodell, das sich mit dem aktuellen Vormarsch des Cloud Computing entwickelt hat. Softwareentwickler können ihre Anwendungen schneller und mit der erforderlichen Flexibilität und Skalierbarkeit bereitstellen und gleichzeitig den Wartungsaufwand für die Infrastruktur sehr gering halten. Diese Vorteile sind im Edge-Computing sehr wünschenswert, da sich ständig ändernde Technologien und Anforderungen schnell umgesetzt werden müssen und die Fluktuation und Heterogenität der Service-Consumer ein wichtiger Faktor ist. Edge-Nodes können jedoch häufig nur einen Bruchteil der Leistung von Cloud-Computing-Infrastruktur bereitstellen, was die Ausführung herkömmlicher FaaS-Plattformen unmöglich macht. In dieser Arbeit stellen wir einen neuen Ansatz für FaaS eine Plattform vor, die von Grund auf unter Berücksichtigung von Edge-Computing- und IoT-Anforderungen entwickelt wurde. Um den Overhead so gering wie möglich zu halten, nutzen wir CoAP als Messaging-Protokoll und Docker, um Applikationen voneinander zu isolieren, während Container wiederverwendet werden, um die bestmögliche Leistung zu erzielen. Wir präsentieren auch eine Proof-of-Concept-Implementierung unseres Designs, die wir mit einem eigenen Benchmarking-Tool getestet haben, und vergleichen unsere Ergebnisse mit Benchmarks von Lean OpenWhisk, einer weiteren FaaS-Plattform für die Edge. Wir stellen fest, dass unsere Plattform Lean OpenWhisk in Bezug auf Latenz und Durchsatz in allen Tests übertreffen kann, Lean OpenWhisk jedoch höhere Erfolgsraten bei einer geringen Anzahl gleichzeitiger Clients aufweist.

Table of Contents

Declaration	I
Abstract	III
Zusammenfassung	IV
List of Figures	VIII
List of Tables	X
List of Listings	XII
I Foundations	3
1 Introduction	5
1.1 Problem Statement	6
1.2 Contributions of this Thesis	6
1.3 Structure of this Thesis	6
2 Background (rename?)	9

2.1	SUMO	9
2.2	Surface Quality	9
2.3	Deep Learning	9
2.4	Citizen Science	9
2.5	Crowdsensing	9
3	Contributions	11
3.1	CycleSense	12
3.1.1	Detecting Near Miss Incidents	12
3.1.2	Evaluation of CycleSense	12
3.1.3	Discussion of CycleSense	12
3.1.4	Summary of CycleSense	12
3.2	CycleQuality	12
3.2.1	Data Processing Pipeline	12
3.2.2	Using Road Surface Quality Information	12
3.2.3	Evaluation of CycleQuality	12
3.2.4	Related Work of CycleQuality	12
3.2.5	Discussion of CycleQuality	12
3.2.6	Summary of CycleQuality	12
3.3	Cyclist Model for SUMO	12

3.3.1	Cycling Behavior in SimRa and SUMO	12
3.3.2	Improving SUMO's Bicycle Simulation	12
3.3.3	Evaluation of the Cyclist Model for SUMO	12
3.3.4	Discussion of the Cyclist Model for SUMO	12
3.3.5	Summary of Bicycle Simulation for SUMO	12
4	Conclusions	13
4.1	Discussion	13
4.2	Summary and Outlook	13
A	Technologies and Concepts of a Larger Context	15
A.1	Finding Consensus in a Distributed System	15
A.2	Detecting Mutual Inconsistency	16
B	Acronyms	19
C	Lexicon	21
D	Listings	23
D.1	Configuration for Node A	23
	Bibliography	25

List of Figures

A.1	Example Setup of Three Machines Agreeing on Values Based on the Paxos	
	Algorithm	16

List of Tables

A.1 Influence of Operations on a Data Item's Version Vector	18
---	----

List of Listings

D.1 Configuration for Node A	23
--	----

Part I

Foundations

Chapter 1

Introduction

The global climate crisis is and will be one of mankind's main challenges in the coming decades. More and more societies worldwide are aware of this problem and search for solutions to adapt to the new changes that comes with the climate change and try to decrease the impact of the global climate crisis. For the latter, the main approach is to reduce greenhouse gas emissions such as CO_2 and NO_x . This is why cities worldwide try to increase the modal share of bicycle traffic and by doing so, decrease the usage of individual motorized transport, which is one of the main greenhouse gas emitters in urban areas (CITATION NEEDED). This, however, puts cyclists, city planners and politicians into a difficult spot. Since cities were developed with the individual motorized transport as the main transportation mode in mind, the traffic infrastructure highly favors the usage of cars. Because of that, cyclists often have a hard time commuting, since they have to cycle on roads that were made for cars, which often puts them into dangerous and stressful situations. Although crash statistics do not convey an increased danger for cyclists (CITATION NEEDED), studies have shown, that the perceived safety of commuting with a bicycle is very low (CITATION NEEDED) and one of the main reasons why people prefer other transportation modes to cycling (CITATION NEEDED). It is up to city planners to improve the traffic infrastructure for cyclists, but it is not easy to pinpoint the most problematic areas, that need to be prioritized for them to be fixed, which

is needed because of the limited resources available to the city planning department. Besides, city planners are bound by city planning codes and rules, which oftentimes also reflect the car-centric approach. This passes the responsibility to the politicians, who need to change the city planning codes and rules and also increase the funds for bicycle infrastructure. For this, however, they need more than anecdotal evidence, that the perceived safety in bicycle traffic is low, so that they can change the legislation and increase funding in favor of bicycle infrastructure.

1.1 Problem Statement

To enable city planners and politicians to do evidence-based decision-making, they need a) to be aware of the problems in bicycle traffic and b) to know where and how to improve the bicycle infrastructure. There are surveys that show that the perceived safety in bicycle traffic is low (MULTIPLE CITATIONS) DO I NEED TO GIVE EXAMPLES HERE FOR SURVEYS?

1.2 Contributions of this Thesis

1.3 Structure of this Thesis

Function-as-a-Service (FaaS) is a cutting-edge service model that has developed with the current advancement of cloud computing. Cloud functions allow custom code to be executed in response to an event. In most cases, developers need only worry about their actual code, as event queuing, underlying infrastructure, dynamic scaling, and dispatching are all handled by the cloud provider [1, 9].

This scalable and flexible event-based programming model is a great fit for IoT event and data processing. Consider as an example a connected button and lightbulb. When the button is pressed it sends an event to a function in the cloud which in turn sends a command to the lamp to turn on the light. The three components are easily connected and only the actual function code would need to be provided. Thanks to managed FaaS, this approach also scales from two devices to thousands of devices without any additional configuration.

Current FaaS platforms do provide these benefits for the IoT, however, using them in this way is inefficient. Sending all events and data to the cloud for processing leads to a high load on the network and high response latency [2, 12]. It is much more efficient to process IoT data closer to their service consumers such as our lightbulb and button, as is the idea of fog computing [3, 11]. Positioned in the same network, our button may send its event to an edge function placed, for example, on a common gateway. This also introduces additional transparency about data movement within the network and alleviates some security concerns about cloud computing [2, 4].

Currently, however, there are no open FaaS platforms that are built specifically for IoT data processing at the edge. State-of-the-art platforms are instead built for powerful cloud hardware, for web-based services, or are proprietary software that is not extensible.

We therefore make the following contributions in this thesis:

1. We discuss the unique challenges of IoT data processing and edge computing and derive requirements for an edge FaaS platform (Chapter 2)
2. We introduce *tinyFaaS*, a novel FaaS platform architecture that addresses the requirements we have identified (Chapter ??)
3. We evaluate *tinyFaaS* through a proof-of-concept prototype and a number of experiments in which we compare it to state-of-the-art FaaS platforms, namely Kubeless and Lean OpenWhisk (Chapter ??)

Chapter 2

Background (rename?)

2.1 SUMO

2.2 Surface Quality

2.3 Deep Learning

2.4 Citizen Science

2.5 Crowdsensing

Chapter 3

Contributions

3.1 CycleSense

3.1.1 Detecting Near Miss Incidents

Preprocessing

Model Architecture

Model Training

3.1.2 Evaluation of CycleSense

Baselines

Metrics

Evaluation Results

3.1.3 Discussion of CycleSense

Impact of Preprocessing & Training Steps

Chapter 4

Conclusions

4.1 Discussion

4.2 Summary and Outlook

Appendix A

Technologies and Concepts of a Larger Context

This appendix chapter contains definitions for technologies and concepts that are mentioned in the thesis, but are not of higher importance for it. Section [A.1](#) introduces a consensus algorithm for distributed systems.

A.1 Finding Consensus in a Distributed System

The Paxos algorithm has first been described by Leslie Lamport in their work about the parliament on the island of Paxos [\[7\]](#). The parliament’s part-time legislators had been able to maintain consistent copies of their records by following the algorithm protocol. While the original work’s main focus has not been the application in computer science, the author simplified their explanations later in [\[6\]](#) and described how Paxos can be used to find consensus in fault-tolerant distributed systems.

Even though the Paxos algorithm cannot mitigate Byzantine failures (see appendix [C](#)), it can mitigate the effects of different processing speeds of participants and reordered, delayed,

or lost messages. The only requirement is that at least half of the participants can somehow communicate.

In Paxos, four different roles exist. The **client** issues requests and waits for responses. Based on the client's request, multiple **proposer** propose a value. All these proposals are processed by the **acceptors** which will agree upon one at the end. Finally, there are the **learners** which guarantee persistence of agreed decisions and respond to client reads. The usual setup used in practice, in which every machine participating fulfills each of the roles besides the client role, is shown in figure A.1.

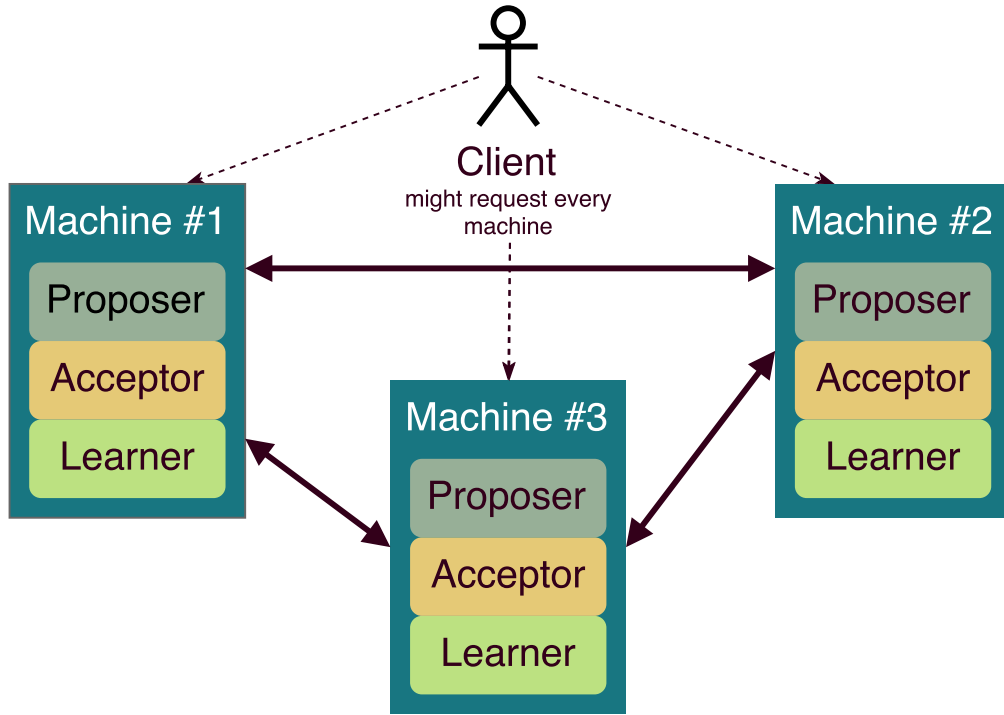


Figure A.1: Example Setup of Three Machines Agreeing on Values Based on the Paxos Algorithm

A.2 Detecting Mutual Inconsistency

Parker et al. claim that a system must ensure the mutual consistency of data copies by applying all changes made to one copy to all others correspondingly [10]. Each time two

copies of the same original data item have a different set of modification applied to them, they become incompatible and this conflict must be detected. However, this is not trivial, because “network partitioning can completely destroy mutual consistency in the worst case”. Nevertheless, Parker et al. state that the efficient detection of conflicts that lead to mutual inconsistency can be done by a concept they call *version vectors*. They define a version vector for an item f as “a sequence of n pairs, where n is the number of sites at which f is stored. The i th pair $(S_i: v_i)$ gives the index of the latest version of f made at site S_i ”. An example vector for an item stored at the sites A, B and D is $\langle A:2, B:4, D:3 \rangle$, which translates in a file that has been modified twice on site A, four times on site B, and thrice on site D.

A set of vectors is “compatible when one vector is at least as large as any other vector in every site component for which they have entries”. Otherwise the vectors conflict and are incompatible. E.g., the two vectors $\langle A:2, B:4, D:3 \rangle$ and $\langle A:4, B:5, D:3 \rangle$ are compatible because the second one dominates the other one. $\langle A:3, B:4, D:3 \rangle$ and $\langle A:2, B:5, D:3 \rangle$ conflict, because the first one indicates that the data item was modified one more time on node A, while the second one indicates one more modification occurred on node B. However, if we add a third vector $\langle A:3, B:5, D:4 \rangle$, no conflict exists anymore, because it dominates the two others. The consequences of an operation performed on a data item for a data item’s vector is depicted in table [A.1](#).

By using version vectors, one can detect version conflicts and initiate (automatic) reconciliation. However, Parker et al. warn that two identical updates made on separate partitions will result in a conflict, even though none is present. Thus, they recommend to additionally check two data items on differences before a conflict is raised in certain applications. Furthermore, the reconciliation is most times not trivial, that’s why tools such as Cassandra delegate this task to the application layer [\[5\]](#).

Operation related to data item	Consequence for vector of data item
Update on site S_i	Increment v_i by one
Delete or rename on site S_i	Keep vector and increment v_i by one, remove data item value
Reconcile version conflict	Set each v_i to maximum v_i from all vectors used for reconciliation. In addition, increment v_i of site that initiated reconciliation by one
Copy to new site	Augment vector to include new site

Table A.1: Influence of Operations on a Data Item's Version Vector

Appendix B

Acronyms

aufs Advanced Multi-Layered Unification Filesystem

AWS Amazon Web Services

CoAP Constrained Application Protocol

EC2 Elastic Compute Cloud

FaaS Function-as-a-Service

HTTP Hypertext Transport Protocol

IaaS Infrastructure-as-a-Service

IoT Internet of Things

IP Internet Protocol

M2M Machine-to-Machine

MQTT Message Queue Telemetry Transport

MQTT-SN MQTT for Sensor Networks

PaaS Platform-as-a-Service

QoS Quality of Service

SaaS Software-as-a-Service

SSL Secure Sockets Layer

TCP Transmission Control Protocol

TLS Transport Layer Security

UDP User Datagram Protocol

URI Unique Resource Identifier

Appendix C

Lexicon

Byzantine failure A malfunction of a component that leads to the distribution of wrong/-conflicting information to other parts of the system is called Byzantine failure [8]. The name is based on the Byzantines Generals Problem, in which three Byzantine generals need to agree on a battle plan while one or more of them might be a traitor trying to confuse the others.

Appendix D

Listings

This is the appendix for code, that does not need to be provided directly inside the thesis.

D.1 Configuration for Node A

Listing D.1: Configuration for Node A

```
{
  "nodeID" : "nodeA",
  "publicKey" : "<public key>",
  "encryptionAlgorithm" : "RSA",
  "machines" : [ "192.168.0.132", "192.168.0.165" ],
  "publisherPort" : 8000,
  "messagePort" : 8010,
  "restPort" : 8080,
  "location" : "52.515249, 13.326476",
  "description" : "Raspberry Pi Cluster in EN 004"
}
```


Bibliography

- [1] I. Baldini et al. “Serverless Computing: Current Trends and Open Problems”. In: *Research Advances in Cloud Computing*. Springer Singapore, 2017, pp. 1–20.
- [2] D. Bermbach et al. “A Research Perspective on Fog Computing”. In: *Service-Oriented Computing – ICSOC 2017 Workshops*. Nov. 2018, pp. 198–210.
- [3] D. Bermbach et al. “Towards Auction-Based Function Placement in Serverless Fog Platforms”. In: *Proceedings of the 2nd IEEE International Conference on Fog Computing 2020 (ICFC 2020)*. 2020.
- [4] F. Bonomi et al. “Fog Computing and its Role in the Internet of Things”. In: *MCC Workshop on Mobile Cloud Computing, MCC@SIGCOMM 2012*. Aug. 2012, pp. 13–15.
- [5] Avinash Lakshman and Prashant Malik. “Cassandra: a decentralized structured storage system”. In: *ACM SIGOPS Operating Systems Review* 44.2 (2010), pp. 35–40.
- [6] Leslie Lamport. “Paxos made simple”. In: *ACM Sigact News* 32.4 (2001), pp. 18–25.
- [7] Leslie Lamport. “The part-time parliament”. In: *ACM Transactions on Computer Systems* 16.2 (1998), pp. 133–169.
- [8] Leslie Lamport, Robert Shostak, and Marshall Pease. “The Byzantine generals problem”. In: *ACM Transactions on Programming Languages and Systems* 4.3 (1982), pp. 382–401.

- [9] G. McGrath and P. R. Brenner. “Serverless Computing: Design, Implementation, and Performance”. In: *2017 IEEE 37th International Conference on Distributed Computing Systems Workshops (ICDCSW)*. June 2017, pp. 405–410.
- [10] D. Stott Parker et al. “Detection of Mutual Inconsistency in Distributed Systems”. In: *IEEE Transactions on Software Engineering* SE-9.3 (1983), pp. 240–247.
- [11] Tobias Pfandzelter and David Bermbach. “IoT Data Processing in the Fog: Functions, Streams, or Batch Processing?” In: *Proceedings of the 1st Workshop on Efficient Data Movement in Fog Computing (DaMove 2019)*. IEEE, June 2019, pp. 201–206.
- [12] B. Zhang et al. “The Cloud Is Not Enough: Saving IoT From the Cloud”. In: *7th USENIX Workshop on Hot Topics in Cloud Computing (HotCloud ’15)*. July 2015.