https://doi.org/10.20378/irb-52238

BAMBERGER BEITRÄGE

ZUR WIRTSCHAFTSINFORMATIK UND ANGEWANDTEN INFORMATIK

ISSN 0937-3349

Nr. 105

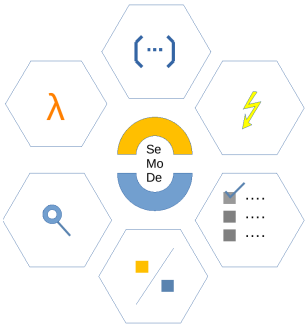
SeMoDe – Simulation and Benchmarking Pipeline for Function as a Service

Johannes Manner

November 2021

FAKULTÄT WIRTSCHAFTSINFORMATIK UND ANGEWANDTE INFORMATIK

OTTO-FRIEDRICH-UNIVERSITÄT BAMBERG



SeMoDe – Simulation and Benchmarking Pipeline for Function as a Service

Johannes Manner

Lehrstuhl für Praktische Informatik, Fakultät WIAI
Otto-Friedrich-Universität Bamberg
An der Weberei 5, 96047 Bamberg

`https://github.com/johannes-manner/SeMoDe`
`https://semode.pi.uni-bamberg.de`

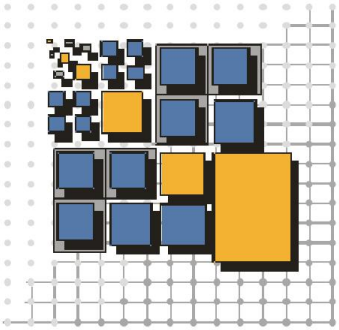
Tool Documentation Version 1.1

Abstract

Cloud computing started with the promise of delivering computing resources elastically at scale, pay per use and on demand self-service to name a few capabilities. In early 2016, Amazon Web Services (AWS) launched a new product called *AWS Lambda* which started the so called *serverless* hype and established a new cloud delivery model, namely Function as a Service (FaaS). FaaS offerings keep the promise of delivering computing resources on demand. They dynamically scale up and down function instances and introduce the most fine-grained billing model across all as-a-service offerings by accounting on a milliseconds basis. Despite this flexibility and the possibility to concentrate on the business functionality, a FaaS user loses operational control. Only a few configuration options remain to tune the functions. The first pay-as-you-go billing model raises new questions for performance-cost trade-offs.
In order to choose a suitable configuration dependent on the use case and get a solid understanding of performance impact of FaaS platforms, **`SeMoDe`** implements a benchmarking and simulation pipeline. It calibrates a physical developer machine, simulates the function in different settings which are comparable to those of cloud offerings and enables a decision guidance to choose an appropriate configuration when deploying it. Based on a Structured Literature Review (SLR) to show the benchmarking and simulation efforts, I suggest a checklist for conducting fair, repeatable and meaningful benchmarks with a focus on documenting the experiments.

Keywords

Cloud Computing, Function as a Service, Benchmarking, Simulation



Distributed Systems Group

Otto-Friedrich Universität Bamberg

An der Weberei 5, 96047 Bamberg, GERMANY

Prof. Dr. rer. nat. Guido Wirtz

http://www.uni-bamberg.de/pi/

Preface

Due to hardware developments, strong application needs and the overwhelming influence of the net in almost all areas, distributed systems have become one of the most important topics for nowadays software industry. Owing to their ever increasing importance for everyday business, distributed systems have high requirements with respect to dependability, robustness and performance. Unfortunately, distribution adds its share to the problems of developing complex software systems. Heterogeneity in both, hardware and software, permanent changes, concurrency, distribution of components and the need for inter-operability between different systems complicate matters. Moreover, new technical aspects like resource management, load balancing and guaranteeing consistent operation in the presence of partial failures and deadlocks put an additional burden onto the developer.
The long-term common goal of our research efforts is the development, implementation and evaluation of methods and tools that support the realization of robust and easy-to-use software for complex systems in general while putting a focus on the problems and issues regarding distributed systems on all levels. Our current research activities are focussed on different aspects centered around Cloud computing, esp. Microservice and Serverless architectures:

- *Integration Testing of Serverless Applications*: Many cloud platform providers offer Function as a Service (FaaS) now which got popular with the introduction of Amazon's AWS Lambda in 2014. These offerings are based on serverless functions whose statelessness helps handle dynamic workloads by scaling them dynamically.Serverless functions are often combined with other services like data storage , e.g., to save the state of the application. The interactions of these services with serverless functions build complex systems. The aim of this project is to support the integration testing process for serverless applications. While it is easy to test single functions in isolation, the emerging behavior caused by the integration of serverless functions with other services needs to be tested. Therefore, the relevant aspects of an application have to be modeled to support the creation of test cases. Coverage criteria are created and their applicability is investigated. Furthermore, the automatic creation of test cases for serverless applications shall be supported.
- *Benchmark and Simulation of Cloud Functions (FaaS)*: The goal of this project is to understand runtime characteristics of the platform as well as characteristics of the deployed cloud functions and take dependent services like database access into consideration when building a local clone of the platform at a developer's machine. These aspects allow to configure cloud functions appropriately to the specified requirements upfront. Furthermore, a simulation and benchmarking tool to conduct repeatable and fair experiments is under development.
- *Architecting Cloud-native Applications*: Cloud-native applications are designed and built to maximally exploit the benefits offered by modern cloud computing. This com-

prises several aspects, such as a fine-grained modular architecture, using existing cloud services instead of custom solutions, exploiting the rapid elasticity of cloud computing, achieving robustness by distributing an application over independent nodes, and finally a faster and more agile development process. The goal of this project is to analyze how the development of such cloud-native applications can be supported and improved with regard to all these aspects. The focus is on the overarching architecture of a cloud-native application, specifically how the individual components are combined and how they interact, rather than focusing on individual components.

- *Universal Cloud-Edge-IoT Orchestration*: The emergence of the Internet of Things (IoT) is a significant development in today's information technology and involves the ability of physical devices to exchange data over networks. Often, the generated data is transferred to the cloud and processed there. Likewise, the cloud may take control of the devices. The ever-increasing number of data-generating IoT devices is creating new challenges that require modifications of the already existing Cloud-edge architectures. This project aims to realize an abstracted, configurable and simplified management of cloud-edge/edge-IoT architectures based on already popular container orchestration platforms like Kubernetes and other platforms and technologies in order to make the use of edge computing easier for even more application areas and users.
- *Visual Programming- and Design-Languages*: The goal of this long-term effort is the utilization of visual metaphors and languages as well as visualization techniques to make design- and programming languages as well as distributed systems more understandable and, hence, more easy-to-use.

More information about our work, i.e., projects, papers and software, is available at our homepage (see above). If you have any questions or suggestions regarding this report or our work in general, don't hesitate to contact me at `guido.wirtz@uni-bamberg.de`

Guido Wirtz

Bamberg, May, 2021

Contents

1 Introduction **1**

2 Related Approaches **3**

3 Checklist for FaaS Benchmarking **11**

4 System Architecture **12**

4.1 Database Layout 12

4.2 Package Diagram 14

5 Setup **16**

5.1 System Setup 16

5.2 Provider Setup 17

5.3 System Startup 17

6 Supported Functionality **17**

6.1 Web Interface 18

6.1.1 Pipeline Configuration 18

6.1.2 Visualizing Execution Data 26

6.2 Benchmarking Cloud Function Platforms 26

6.2.1 Implementing a Function under Benchmark 26

6.2.2 Benchmarking Modes 26

6.2.3 Submitting Requests based on Arbitrary Load Pattern 28

6.3 Simulating Local Cloud Function Executions 30

6.3.1 Calibrating Hardware 30

6.3.2 Simulation based on Calibration 31

6.4 REST API 31

6.4.1 REST API Security 31

6.4.2 Exposed REST API Endpoints 32

6.5 CLI Features 32

7 Conclusion and Future Work **33**

Acknowledgment **38**

A User Management **38**

List of previous University of Bamberg reports **39**

List of Figures

1 SLR conducted at dblp on 23rd of June 2021. 3
2 Number of experiments per provider. 8
3 Overall System Architecture of SeMoDe 12
4 Database Layout of SeMoDe 13
5 Package Diagram of SeMoDe 15
6 Setups Page to See Existing Setups and Create Them 18
7 Benchmark Configuration Page I 19
8 Benchmark Configuration Page II 21
9 Calibration Configuration Page I 23
10 Calibration Configuration Page II 24
11 Calibration Configuration Page III 25
12 Benchmark Publicly Visible Page 27
13 User and Platform Perceived Performance (based on Figure 1 in [1]) 28
14 CPU performance computed by LINPACK at different CPU Quotas (based on Figure 1 in [2]) 30
15 Open API Specification (OAS) for exposed REST API. 32
16 Users Page for Administrators to Change Roles and Passwords. 38

List of Tables

1 Research conducted by using SeMoDe. 2

2 SLRs of the conducted SLR. 4

3 FaaS research context and related papers to the intended aims of SeMoDe. . . 5

4 Papers Related to SeMoDe Research Goals. 7

Listings

1 Centerpiece of the `BenchmarkExecutor` Class 29

2 Start `SeMoDe` as CLI application 33

Abbreviations

ACM	Association for Computing Machinery
ARN	Amazon Resource Name
AWS	Amazon Web Services
AZF	Azure Functions
DTO	Data Transfer Object
FaaS	Function as a Service
GCF	Google Cloud Functions
GFLOPS	Giga FLoating point Operations Per Second
HPC	High Performance Computing
IaaS	Infrastructure as a Service
IaC	Infrastructure as Code
IAM	Identity and Access Management
IEEE	Institute of Electrical and Electronics Engineers
JIT	Just In Time
JRE	Java Runtime Environment
JPA	Java Persistence API
JWT	JSON Web Token
NIST	National (US) Institute of Standard and Technology
OAS	Open API Specification
ORM	Object Relational Mapping
OS	Operating System
PaaS	Platform as a Service
RTT	Round Trip Time
SLR	Structured Literature Review

1 Introduction

FaaS, a concept where ephemeral and stateless cloud functions run only upon request, is the next evolution in cloud computing [3, 4]. It enhances the cloud computing capabilities [5] defined by the National (US) Institute of Standard and Technology (NIST) to prior service models like Platform as a Service (PaaS) or even Infrastructure as a Service (IaaS). *Broad network access* and *resource pooling* characteristics do not significantly change when comparing FaaS with PaaS. Whereas *on-demand self-service* results in a provisioning of cloud function instances for every single request which also positively influences the *rapid elasticity* characteristic. Most importantly, the billing model is leading to a *measured service* in 100 millisecond chunks. In December 2020, AWS Lambda[1], the first commercial offering of cloud functions, recently announced the most fine-grained billing model in a granularity of milliseconds[2].

The motivation for this report and the ongoing research project[3] is to shed some light onto cloud function platforms' inner workings and their runtime behavior. The only possible way to understand the improved cloud computing characteristics by FaaS is to deploy different functions in a production environments and assess the results. This procedure is often time-consuming and costly and the results of these benchmarks are seldom *repeatable* nor *portable* to other functions [6]. Also other benchmarking characteristics [7, 8] like *simplicity* or *efficiency* are hard to achieve when designing a benchmark but even more challenging when other researchers want to assess and repeat the experiments since a lot of publications try to "fool the masses with irreproducible results"[4].

Nevertheless, benchmarks are an important step when developers want to deploy a function. They decide on the configuration of a function based on the benchmark data, but the functions deployed in production and the benchmark functions often have different characteristics.

Therefore, the second aspect of our research prototype is the simulation of cloud function behavior on a local machine. This removes the burden of costly experiments in the cloud and enables an understanding of the runtime behavior of the cloud functions. The tool provides a simulation pipeline with the assistance of collected data from selected benchmarks and enables developers to assess their function under development, before deploying it, to find the best configuration for their requirements, e.g. lowest cost or shortest execution time. To do so, it collects data from cloud providers to understand their platform implementation and the impact of different resource configurations on the function execution first. In a second step, the prototype executes a similar calibration on a developer machine to relate the local performance to the cloud performance. Based on this data, local function executions can then predict the execution behavior of a deployed function before deploying it.

The first research question motivates the SLR, while the second builds upon the results and condenses best practices for documenting experiments. Furthermore, the following three statements highlight the contribution `SeMoDe` makes to answer the research questions.

[1] `https://aws.amazon.com/de/lambda/`

[2] `https://aws.amazon.com/de/blogs/aws/new-for-aws-lambda-1ms-billing-granularity-adds-cost-savings/`

[3] `https://www.researchgate.net/project/Serverless-Computing-Support`

[4] Title of the keynote from Lorena Barba at the 2021 edition of IEEE International Parallel and Distributed Processing Symposium, `https://www.youtube.com/watch?v=R2-GuH-6VFU`.

RQ1 - Which tools and experiments exist for benchmarking FaaS platforms and performing simulations?

RQ2 - How should a FaaS experiment be documented and which items are necessary for a convincing data evaluation?

S1 - `SeMoDe` implements a novel concept to reach dev-prod parity for cloud functions based on its benchmarking and simulation features.

S2 - `SeMoDe` enables reproducibility and documentation of experiments through its design and versioning of experimental settings.

S3 - `SeMoDe` invites other researchers to collaborate, reconstruct already conducted experiments and share new benchmarking results based on its public interface.

These statements are explained in detail, when we look at `SeMoDe`'s capabilities and the design decisions. Related approaches are discussed in Section 2 based on a SLR answering **RQ1**. Lessons learned from the SLR are summarized in a checklist to emphasize important aspects for building a fair, repeatable and meaningful FaaS benchmark (see **RQ2**). Section 4 introduces the overall architecture of the prototype by discussing the different components and tools, the database and package layout. The subsequent Section describes the setup and configuration of related provider tools which are necessary to execute benchmarks on public cloud provider hosted FaaS platforms. Section 6 describes the features the prototype implements, especially the configuration via the web interface and the already mentioned benchmarking and simulation pipeline. We conclude the technical report with Future Work considerations.

At the time of writing this technical report, `SeMoDe` contributed to the evaluation or practical part of five papers. I contributed to all these papers as the main author. They are ordered anti-chronological in Table 1.

Table 1: Research conducted by using `SeMoDe`.

Title	**Venue**	**Presented**	**Ref**
Optimizing Cloud Function Configuration via Local Simulations	CLOUD	2021	[9]
Why Many Benchmarks Might Be Compromised	SOSE	2021	[2]
Impact of Application Load in Function as a Service	SummerSoC	2019	[10]
Cold Start Influencing Factors in Function as a Service	WoSC	2018	[1]
Troubeshooting Serverless Functions: A Combined Monitoring and Debugging Approach	SummerSoC	2018	[11]

2 Related Approaches

A lot of FaaS research was done conducting benchmarks and investigating special properties of the platforms as well as finding ways to optimize the cloud functions runtime behavior. In this Section, I will highlight some related respectively competing approaches to the research prototype described in this report.

To get a solid basis and a repeatable literature design, I did a SLR based on the guidelines of Kitchenham and Charters [18]. After defining a short review protocol[5], I picked the dblp computer science bibliography[6] as a source since it includes work from journals and conference proceedings from different publishers like Association for Computing Machinery (ACM) and Institute of Electrical and Electronics Engineers (IEEE).

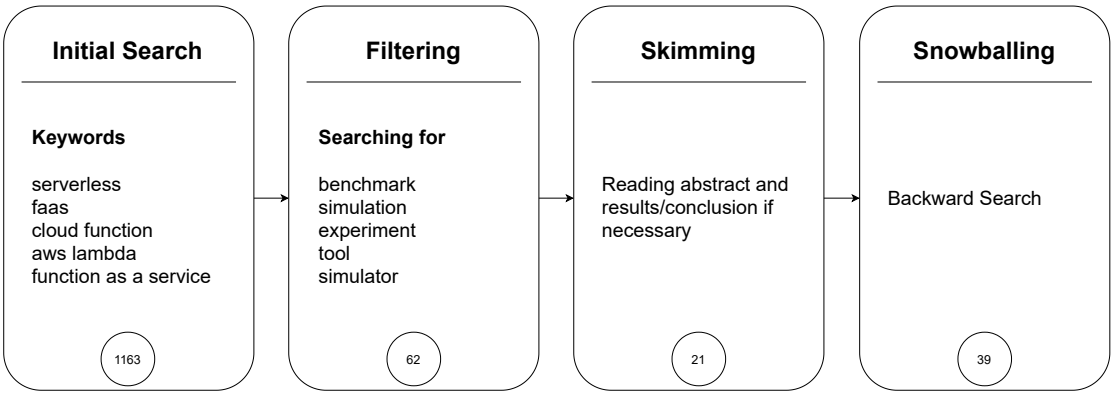


Figure 1: SLR conducted at dblp on 23rd of June 2021.

Figure 1 shows the SLR process. Since the advanced search at dblp was disfunctional when combining several search terms with an OR operator, I decided to conduct multiple searches with the keywords presented in the initial search phase. I used a set of generic keywords to get a broad basis for the further steps. This resulted in 1240 entries. To handle the literature, JabRef[7] was used which helped with removing duplicates in this first phase. Since this Section is about finding competing approaches to the already published papers, I excluded the publications mentioned in Table 1. The remaining 1163 entries were then filtered by the search feature of JabRef. Similarly to the search on dblp, I also decided to perform a single search for *benchmark*, *simulation*, *experiment*, *tool* and *simulator*. The filtering keywords were chosen due to the focus on understanding benchmark and simulation approaches in FaaS as well as looking at experiments and tools. The filtering process was done on the title and other metainformation like the conference or journal name. After removing five duplicates, I ended up with 62 papers which I skimmed in the next phase. For most of them, it was sufficient to read the abstract to evaluate their relevance. To remove doubts on ambiguous papers, I read the research questions and the results part to make a well justified decision. After the skimming phase, 21 papers were left which I read in detail while answering the questions of the review protocol. Since the search was only done on dblp and the keywords were quite generic for the initial search and the filtering process, I decided, when discussing the SLR process with colleagues, to make a backward

[5] Protocol plus results of the SLR, `https://github.com/johannes-manner/SeMoDe/files/7562710/20211118_slr.xlsx`.
[6] `https://dblp.uni-trier.de/`
[7] `https://www.jabref.org/`

search to identify further relevant research. The snowballing process was done for all 21 papers in the third step and also the papers which were identified during snowballing. After finishing this recursive snowballing process, 39 papers were part of the final set. The following Tables show the literature set by categorizing them in SLRs (Table 2), papers which are not directly related to experiments on public cloud providers (Table 3), competing approaches, and insights which are important for the prototype design and the improvement of empirical FaaS research (Table 4).

Table 2: SLRs of the conducted SLR.

Ref	Authors	Title	Year
[19]	Yussupov et al.	A Systematic Mapping Study on Engineering Function-as-a-Service Platforms and Tools	2019
[20]	Scheuner and Leitner	Function-as-a-Service performance evaluation: A multi-vocal literature review	2020
[6]	Kuhlenkamp and Werner	Benchmarking FaaS Platforms: Call for Community Participation	2018

Table 2 includes SLRs, which are not discussed in detail here, but each addresses a specific aspect of the FaaS research domain. Yussupov and others [19] motivate the work on a consistent benchmark and simulation framework due to their SLR being focused on different platforms and tools. Performance aspects are discussed in Scheuner's and Leitner's work [20] where they included 112 sources from academic and gray literature. Especially interesting is the design of different experiments found in literature and their impacts on the data evaluation and therefore on the results. Kuhlenkamp and Werner [6] motivated the community to participate in the benchmarking process by revealing issues about the reproducibility of conducted research. The authors stated in their RQ3 that only 3 out of 26 experiments were reproducible based on the provided information. This motivates my statement **S2** to enable other researchers to reproduce the experiments conducted with `SeMoDe` by looking at the experiment documentation.

Table 3 contains the publications, I classified as related to my empirical FaaS research but not directly supporting my efforts in building the benchmark and simulation pipeline intended by `SeMoDe`. Nevertheless, the contributions of these papers are important to stress the context and related research areas. The first entry [21] is a vision paper about how a benchmark in the FaaS area should look like with a focus on real world experiments to overcome microbenchmarks. Mohanty and others [22] evaluated the open source platforms Fission[8], Kubeless[9] and OpenFaaS[10]. They focused on testing how concurrent users and the autoscaling property, which is based on Kubernetes Horizontal Pod Autoscaler, influence the number of running function instances. Since open source solutions based on K8s are not in the focus of this work, the paper is not included in Table 4 for further investigation. The same holds true for the edge computing frameworks, where the first work [23] used AWS Greengrass[11] and Azure IoT Edge[12] for latency measurements and cost comparisons. The second publication [24] implemented an extension for CloudSim [35], a popular simulator for

[8] `https://fission.io/`
[9] `https://kubeless.io/`
[10] `https://www.openfaas.com/`
[11] `https://aws.amazon.com/greengrass/`
[12] `https://azure.microsoft.com/en-us/services/iot-edge/`

Table 3: FaaS research context and related papers to the intended aims of `SeMoDe`.

Ref	Authors	Title	Year
[21]	van Eyk et al.	Beyond Microbenchmarks: The SPEC-RG Vision for a Comprehensive Serverless Benchmark	2020
[22]	Mohanty et al.	An Evaluation of Open Source Serverless Computing Frameworks	2018
[23]	Das et al.	EdgeBench: Benchmarking Edge Computing Platforms	2018
[24]	Jeon et al.	A CloudSim-Extension for Simulating Distributed Functions-as-a-Service	2019
[25]	Kritikos and Skrzypek	Simulation-as-a-Service with Serverless Computing	2019
[26]	Bardsley et al.	Serverless Performance and Optimization Strategies	2018
[27]	Gan et al.	An Open-Source Benchmark Suite for Microservices and Their Hardware-Software Implications for Cloud & Edge Systems	2019
[28]	Malla and Christensen	HPC in the cloud: Performance comparison of function as a service (FaaS) vs infrastructure as a service (IaaS)	2019
[29]	Malawski et al.	Serverless execution of scientific workflows: Experiments with HyperFlow, AWS Lambda and Google Cloud Functions	2020
[30]	Giménez-Alventosa et al.	A framework and a performance assessment for serverless MapReduce on AWS Lambda	2019
[31]	Quaresma et al.	Validation of a simulation model for FaaS performance benchmarking using predictive validation	2021
[32]	Gias and Casale	COCOA: Cold Start Aware Capacity Planning for Function-as-a-Service Platforms	2020
[33]	McGrath and Brenner	Serverless Computing: Design, Implementation, and Performance	2017
[34]	Ustiugov et al.	Benchmarking, Analysis, and Optimization of Serverless Function Snapshots	2021

cloud offerings, with the aim to support a distributed FaaS architecture comprised of cloud, fog and edge.
Another subarea are use case implementations for using FaaS platforms as execution environment for simulations [25], cloud functions as part of an e-commerce system [26], building microservice architectures with predictable performance [27] or checking the High Performance Computing (HPC) capabilities of cloud function offerings compared to IaaS [28] or against each other [29]. A map reduce use case was implemented by Giménez-Alventosa and others [30]. Of particular interest is their data analysis where they grouped the invocations of the functions by the executing VM. In the resulting histogram, there were deviations of up to approximately 30% in execution time. Since they missed to also record the physical machine's information, first of all the CPU specifications, the reason for this huge deviation remains unanswered. When relating these measures with other experiments [14, 15, 16], one explanation could be the different physical machines used for deploying the VMs and executing the cloud function. These insights were one reason to include the VM identification and the hardware specification of the executing machine in my checklist and also in the function trigger response discussed in Section 6.2.1.
The last group of papers optimized some aspects of FaaS platforms by making a custom implementation or configuration. These aspects are also out of scope for implementing a benchmark and simulation pipeline for assessing function performance and characteristics on public FaaS platform offerings, but highlight shortcomings in today's FaaS offerings. Quaresma and others [31] built a simulator to predict their prior introduced suppression of garbage collection during function execution and compare this to invocations on public providers. The results presented there are hard to interpret since a lot of deployment and configuration details were missing. Solving the startup of *unnecessary* instances in over-provisioning scenarios are researched by Gias and Casale [32]. They implemented a queuing based approach for on premise platforms which breaks the scaling on demand principle of FaaS. The next work [33] implemented a prototype in .NET deployed on Azure's cloud platform, compared it to public offerings and showed some benefits especially for throughput (executions per second).
Noteworthy is the effort of Ustiugov and others [34]. Their major research goal is to provide a serverless open source playground[13] for experimentation on various layers of the system stack like Firecracker hypervisor[14], which was open-sourced by AWS and used for their production AWS Lambda environment.

`SeMoDe` tackles public cloud provider offerings and tries to understand their design, scheduling and quality of service. Since the remaining 22 publications listed in Table 4 are more closely related to these efforts, they are discussed in more detail. Figure 2 shows a distribution of which platforms were investigated in the selected papers. Commonly, a few functions were often used to research different aspects of FaaS namely matrix multiplication (5), hello world functions (3), prime number search (3), integer factorization (2) and LINPACK (2).

My ordering of the platforms is similar to Scheuner and Leitner [20] where AWS Lambda is the most investigated platformed followed by Azure Functions (AZF), Google Cloud Functions (GCF) and lastly IBM OpenWhisk. Important for repeatable research is to enable other researcher to reproduce the experiments. Only 17 out of the 22 publications have open sourced their prototypes.

One of the first benchmark papers was published in 2018 [36] which includes all major cloud

[13] `https://github.com/ease-lab/vhive`
[14] `https://firecracker-microvm.github.io/`

Table 4: Papers Related to SeMoDe Research Goals.

Ref	Authors	Title	Year
[36]	Back and Andrikopoulos	Using a Microbenchmark to Compare Function as a Service Solutions	2018
[37]	Pellegrini et al.	Function-as-a-Service Benchmarking Framework	2019
[38]	Somu et al.	PanOpticon: A Comprehensive Benchmarking Tool for Serverless Applications	2020
[39]	Jackson and Clynch	An Investigation of the Impact of Language Runtime on the Performance and Cost of Serverless Functions	2018
[40]	Kuntsevich et al.	Demo Abstract: A Distributed Analysis and Benchmarking Framework for Apache OpenWhisk Serverless Platform	2018
[41]	Lee et al.	Evaluation of Production Serverless Computing Environments	2018
[42]	Martins et al.	Benchmarking Serverless Computing Platforms	2020
[43]	Kim and Lee	FunctionBench: A Suite of Workloads for Serverless Cloud Function Service	2019
[44]	Kim and Lee	Practical Cloud Workloads for Serverless FaaS	2019
[45]	Copik et al.	SeBS: A Serverless Benchmark Suite for Function-as-a-Service Computing	2020
[46]	Grambow et al.	BeFaaS: An Application-Centric Benchmarking Framework for FaaS Platforms	2021
[47]	Bortolini and Obelheiro	Investigating Performance and Cost in Function-as-a-Service Platforms	2019
[48]	Wang et al.	Peeking Behind the Curtains of Serverless Platforms	2018
[49]	Lloyd et al.	Serverless Computing: An Investigation of Factors Influencing Microservice Performance	2018
[50]	Pons and López	Benchmarking Parallelism in FaaS Platforms	2020
[51]	Kuhlenkamp at al.	Benchmarking Elasticity of FaaS Platforms as a Foundation for Objective-driven Design of Serverless Applications	2020
[52]	Maissen et al.	FaaSdom: a benchmark suite for serverless computing	2020
[53]	Malawski et al.	Benchmarking Heterogeneous Cloud Functions	2017
[54]	Pawlik et al.	Performance evaluation of parallel cloud functions	2018
[55]	Figiela et al.	Performance evaluation of heterogeneous cloud functions	2018
[56]	Yu et al.	Characterizing serverless platforms with serverlessbench	2020
[57]	Mahmoudi and Khazaei	SimFaaS: A Performance Simulator for Serverless Computing Platforms	2021

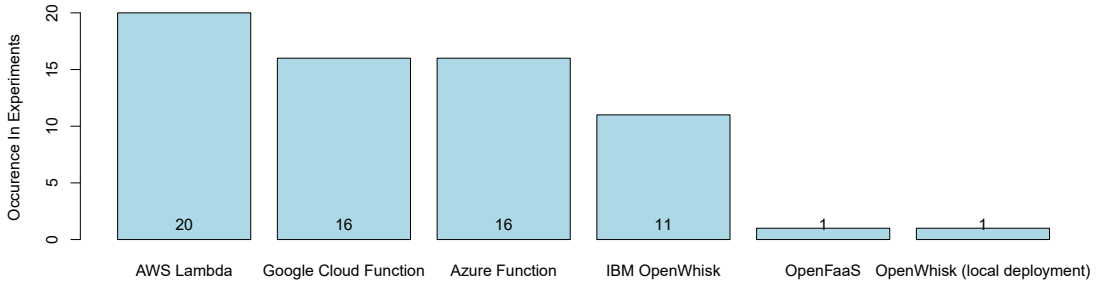


Figure 2: Number of experiments per provider.

providers namely AWS Lambda, GCF, AZF and IBM OpenWhisk. Their motivation was to understand the maturity of different platforms as well as the relation of performance, function settings and cost. The individual aspects are answered, but their inter-relations were not approached in detail. Pellegrini and others [37] focused on measuring routing properties which was done via a proxy function to understand the inner platform routing. Since the implementation is not open sourced and due to some missing information about the configuration of the experiment, the paper is only partly interpretable. Another work [38] implements a tool for deploying function and other components to different providers. The function deployment can also, as described in the paper, be done by using the serverless framework[15] which is also used by other papers included in this SLR, like [53, 36, 42, 51]. The benefits over generic Infrastructure as Code (IaC) tools like Terraform[16] or provider specific like AWS CloudFormation[17] are not clear since the preliminary results are not discussed in detail.

Different languages and their effects on execution time were investigated by Jackson and Clynch [39] for AWS Lambda and AZF. What is missing in this paper is the memory setting for AWS Lambda, which has an influence on the execution time [9]. Another language comparison is done by Kuntsevich and others [40] tackling a locally hosted OpenWhisk installation. Due to the nature of the paper being a demo, they only included first results on the execution time behavior for concurrent requests. Another concurrency study [41] was motivated to compare FaaS to an IaaS VM offering. A data intensive application was used to show the fit of FaaS for distributed computing where also other researchers see the benefits of this computing model like Jonas and others [58]. A lot of execution data and its analysis was published by Martins and others [42] which contains some noise. Especially the Round Trip Time (RTT) is included in the data without clarification. Furthermore, the HelloWorld use cases in different languages are limited in their interpretation since there are constellations where a function running in Java is faster than the JS counterpart as shown in [1]. Therefore, the use case respectively function characteristic is crucial to choose the right programming language and platform. One experiment in Martin's work is comparable to other research, e.g. [59, 1, 15, 60, 61, 9], where a fibonacci function was executed at different memory settings on different providers to understand the resource scaling strategies.

As argued before [9], microbenchmarking can unravel some platform mystery when designing

[15] `https://www.serverless.com/`
[16] `https://www.terraform.io/`
[17] `https://aws.amazon.com/cloudformation/`

experiments in a way so that a single aspect is isolated. For that reason, Kim and Lee [43, 44] published a benchmark suite comprised of microbenchmarking and application level benchmarks as a foundation for experiments to discuss about common cloud functions and applications in research. Nevertheless, they argued that microbenchmarking is important but not sufficient for the requirements real world use cases present. Authors of other real world experiments like SeBS [45] and BeFaaS [46] enforced that argument but are aware that real world use cases are often not that precise in their conclusions due to the noise of the collected data. However, they share insights on the adaptation of the technology. SeBS [45] concludes that IO bound functions in general do not profit from cloud function execution model due to the double billing problem [62]. They also included a cost analysis and compared their real world examples to IaaS solutions. Despite beeing also real world conform by implementing a webshop and traffic light use case, BeFaaS [46] has a different focus. They focus on federated workloads where their traffic light application is comprised of a cloud and edge layer. The latter is implemented by using their tinyFaaS [63] system. Some information like the configurations for the cloud functions is missing and, therefore, the comparison of different providers is limited.

An important aspect in FaaS research is to understand the resource scaling of the providers. AWS Lambda claims that it "allocates CPU power in proportion to the amount of memory configured"[18]. One study confirmed this statement in an ideal world with CPU intensive functions [47] which corresponds to our findings [9]. Resource scaling and runtime prediction is not that deterministic on the other platforms namely GCF and IBM OpenWhisk. Since we already know that the physical host executing the function determines the resource assignment and ultimately the execution time, Wang and others [48] found a way to uniquely identify VMs when executing the cloud functions. Their approach differs from the VM uptime approach suggested by Lloyd and others [49] and is collision free. Back in 2018, they recognized performance drops when a lot of function request were made concurrently, leading to a placement strategy where AWS Lambda executes multiple cloud function on the same VM. Therefore, performance isolation was not guaranteed. In 2020, Pons and López [50] also tackled the request parallelism in FaaS, however, they did not recognize a noisy neighbor effect in their data. A hint that the AWS Lambda platform improved in this property which is also supported by the AWS Security[19] documentation where the software stack AWS Lambda is running on is depicted. They also discovered in their work that the different scheduling strategies, how requests are distributed to instances, impact the fit of different platforms for application classes like IO bound functions and AZF. Kuhlenkamp and others [51] were interested in another scaling aspect. They created five workloads with different characteristics to understand the scaling, i.e. the creation of new cloud function instances. To distinguish between the execution time, understanding the network overhead, request scheduling etc. at the platforms, they used the same measurement points as introduced in Figure 13. Since they only executed a single function (prime number search) on a single memory setting, their relation to SeMoDe - finding the optimal configuration for a function - is limited, but related to understanding the impact of the application workload on the number of instances [10].

As stated, deploying cloud functions can be done via custom tools like the serverless framework or generic IaC tools like Terraform. One design decision of SeMoDe is to not rely on these tools and implement the deployment and monitoring of the function in one compre-

[18] `https://docs.aws.amazon.com/lambda/latest/dg/configuration-function-common.html`
[19] `https://docs.aws.amazon.com/whitepapers/latest/security-overview-aws-lambda/security-overview-aws-lambda.html`

hensive tool. This gives the user the freedom to investigate special hypotheses and using the deployment information for starting the benchmark and getting the logs from the different providers. The same approach was also chosen by Maissen and others [52] where they showed the steps which are needed for deploying the cloud functions to the different platforms, see Figure 4 in their publication. In SeMoDe, I implemented the same deployment steps. They also documented different processors and cold start times by executing the cloud functions in various regions. Their application is also able to estimate the price of the executed benchmarks based on the pricing schemes of the providers. Another set of publications [53, 54, 55] were done by a research group from AGH University of Science and Technology in Poland. Especially their hypotheses based approach inspired our work [1] as well as the LINPACK use case, where *pure* performance of the cloud function based on GFLOPS is measured which is used in our publications, too [2, 9]. They raised questions about the computational performance proportional to function size, the network performance, overhead introduced by network and scheduling, reusage of application instances and their recycling time as well as the heterogeneous hardware. When they conducted their research on AWS Lambda back in 2017, the service was limited to a single vCPU and 1536MB memory, but they already found a linear relationship as stated by the AWS Lambda documentation. For other providers, they did not find that relation. When designing the functionality of SeMoDe in 2018, this was a major reason to focus on AWS Lambda since the linear scaling of resources is predictable and a prerequisite to build repeatable benchmarks and simulation models. In an ideal world, for embarrassingly parallel computation, the service implementation of AWS Lambda leads to a situation of constant cost when raising the memory setting of the function as depicted in their cost calculation in Figure 12 in [55]. Or to say it differently, a user gets the same portion of computing power over time for the same price as also confirmed in our work [9]. Similar to Maissen, they also found different CPU models executing the cloud function, which could also explain the scatter plot in Figure 5 in [55] (respectively Figure 4 in [53]), where two performance ranges for LINPACK were visible. The last included benchmarking paper in the SLR is another workbench, called ServerlessBench [56]. They included an experiment with an multi-threaded application compared to parallel function instances. Since the memory configuration of the different cloud functions included in the plots are missing, the results are only partly interpretable. Their second implication is that splitting parallelizable parts of a function into several concurrently executed functions is use case dependent. We showed [9] that an efficient fork join implementation of a prime number search can also benefit from multi-threading. Nevertheless, their focus on splitting functions dependent on their resource needs, i.e. IO-bound function and CPU-bound functions, is interesting for further investigation.

Most of the papers included in the SLR are benchmarking papers. Since the aim of SeMoDe is to build a simulator based on benchmarking data, we explicitly included *simulation* as a search term. Only two publications were identified with a strong focus on simulating the FaaS offering. The first simulation paper was classified as out of scope of `SeMoDe`'s capabilites by extending CloudSim [24]. The second published an interesting approach and first ideas for a simulator focused on FaaS which simulates the scheduling, initialization etc. of new instances were introduced by Mahmoudi and Khazaei [57]. They included a short evaluation of their ideas and compared the results to platform data from AWS Lambda. They especially focused on scaling strategies, i.e. scale by request, a fixed concurrency threshold and metrics based scaling. `SimFaaS` predicts cold start probability, average response time, rejection of request similar to our prior work [10]. The configuration of the function and its tuning was out of scope of this work.

3 Checklist for FaaS Benchmarking

In the previous Section, I compared the different benchmarking approaches to argue which aspects are important for a reproducible and interpretable experiment design. `SeMoDe` helps in this case, since a lot of data, which is necessary to draw strong conclusions, is collected due to the design and implementation of the prototype. A thorough documentation of open source research tools is the enabler for interpreting results and reproducing them [64] which motivates this work in general and the following checklist. Each item is attached to some references, where details about the data extraction on the respective platform or the methodology is described in detail. As stated when writing about the flaws of some experiments, this checklist tries to help an experimenter to build a multi-dimensional data set for FaaS benchmarking research.

Physical Machine Configuration - Obviously the physical machine and its performance influence measurements. A lot of experimenters do not include this information in their experiment description. The suggestion is to document at least the CPU model, the model number and the Operating System (OS) [52, 53]. Also check the machine's performance where components of your benchmarking process are executed [2, 9].

VM/Container Identification - In the FaaS research area, most of the platforms are designed with some virtualization layers. Note the executing VM or the container configuration and use collision free approaches to identify the executing unit [48].

Function Configuration - The function configuration is often omitted in papers as stated in the SLR. Since the resource scaling is determined by the configuration[20], i.e. the memory setting which influences the resource scaling at AWS Lambda and GCF, this information is vital for classifying the results [9, 47].

Programming Language - The programming language has a big influence on the execution time of the function when thinking about the Just In Time (JIT) compiler of Java compared to interpreted languages [1, 39, 40, 42].

Data Measurement Procedure - For the interpretation of the data, the measurement methodology is important to state in the experiment like Figure 13 shows [1, 51].

Workload - Describe your workload and publish a distribution of it. This gives a first hint on the instance parallelism the FaaS platform has to handle [10, 50, 51].

Cold/Warm Distinction - One of the major benefits of FaaS is scaling on demand. Since this results in function instance creation and cold starts, it is important to state whether the functions faced a cold start or whether the function instance was reused [1, 49].

Multithreaded Implementation - Due to the per-request execution model and the most fine-grained billing model on millisecond basis, the implementation of cloud functions directly influences the application/function characteristics which is different to e.g. PaaS where single functionality in the scope of a cloud function is hidden within a larger application/microservice. Therefore, the multithreading aspect when optimizing performance and costs of cloud function is important to consider for FaaS experiments

[20] AZF uses a different approach to select the configuration for the user when executing the function.

when using multi-core configurations [9, 56]. This is different to long running applications in a PaaS area where multi-threaded implementations of pieces of functionality are suspended for code readability and maintenance which is also addressed by static code quality tools[21].

4 System Architecture

SeMoDe is a full stack app implemented in Java. PostgreSQL[22] is used as relational database solution, Spring Boot[23] as framework for the implementation of our application, Thymeleaf[24] as a server side templating engine and Open API Specification[25] to describe the implemented REST API. Figure 3[26] shows the different layers of the app and references the related Sections of this report in parentheses.

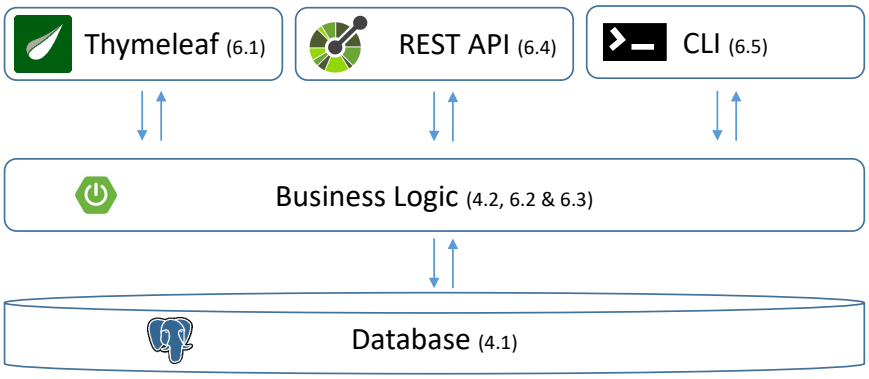


Figure 3: Overall System Architecture of SeMoDe

4.1 Database Layout

The database layout shown in Figure 4 visualizes all tables and their attributes. Primary keys are the first attribute of the specific entity layouted in bold. The italic and bold-faced attributes are foreign keys. The cardinalities of the relationships are directed as implemented in the Java Persistence API (JPA) mapping. They can be interpreted as follows:
A *setup config* is associated with exactly one *user*, whereas a *user* can have 0 to N *setup configs*.

The central table is `setup_config` which is associated with a `user`. Normally, new users get the role `USER` but can request a role change by an administrator of the application. Only the owner (associated user) or users with the role `ADMIN` can see and change the setup configuration and all related entities. These are the only roles currently defined.

[21] https://spotbugs.readthedocs.io/en/stable/bugDescriptions.html#multithreaded-correctness-mt-correctness
[22] https://www.postgresql.org/
[23] https://spring.io/projects/spring-boot
[24] https://www.thymeleaf.org/
[25] https://swagger.io/specification/
[26] Logos for Thymeleaf, OAS, Spring Boot, and PostgreSQL are from the corresponding tool sites.

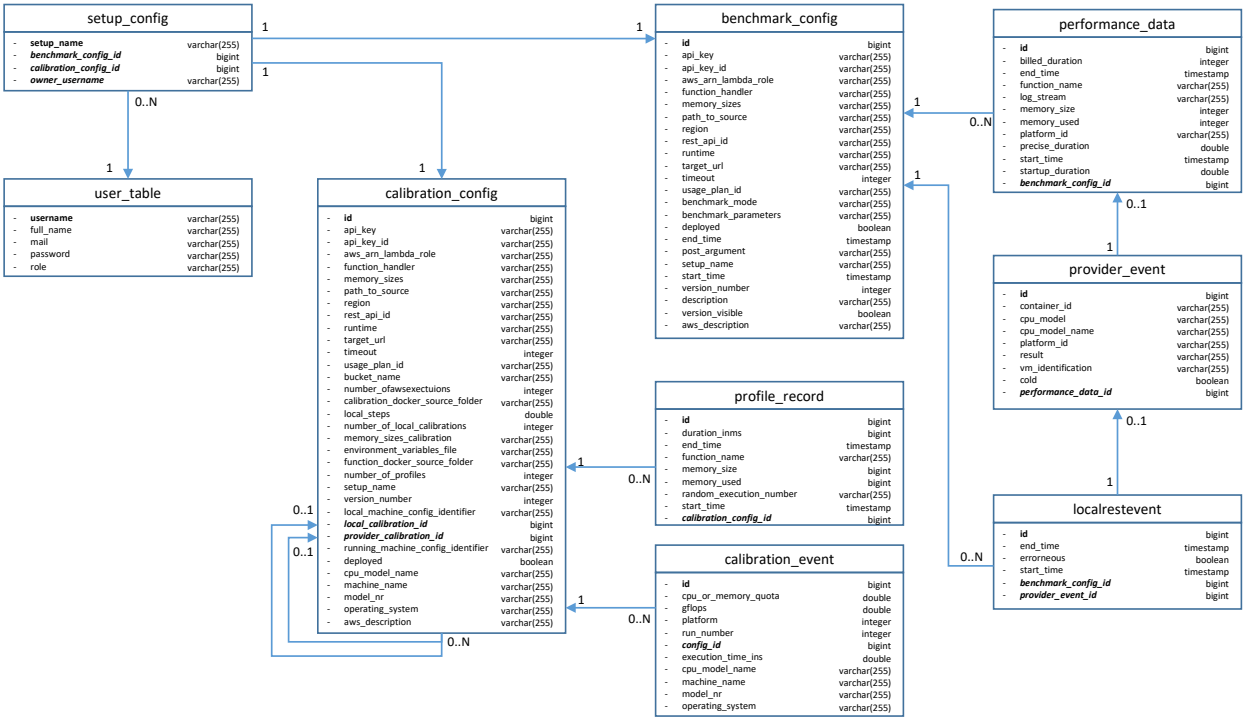


Figure 4: Database Layout of `SeMoDe`

Furthermore, the setup configuration is associated with a single benchmarking and calibration configuration. Only the currently active configurations are associated and only these are editable. If a change occurs, the tool increments the version number. Both tables, `calibration_config` and `benchmark_config`, contain a column `setup_name` which enables a bidirectional relationsship. This is not implemented in JPA but helps in optimizing retrieval operations for a single setup when using native queries, e.g. getting all versions for a specific setup by only querying a single table.

A calibration configuration further has two self-references. One is named `local_calibration` and the other is named `provider_calibration`. Both of them are needed to compare different virtualized environments with each other, a more detailed explanation can be found in [9]. To enable this comparison, `calibration_event`s are needed, which store the Giga FLoating point Operations Per Second (GFLOPS) information for specific memory settings or CPU quotas of the local environment. Sometimes misconfigured systems can be identified where the performance scaling has an insufficient linear correlation when increasing CPU resources as [2] shows and, therefore, cannot be used for the simulation approach proposed. `profile_record`s are the result of simulations and a decision basis for configuring cloud function locally.

Conceptually, the `CalibrationConfig` contains a few objects in our implementation to encapsulate specific aspects, but the `@Embedded` JPA annotation is used to store the attributes of these embedded classes in the same table for sake of simplicity, readability and speed of database retrieval of the experiment configuration.

The `BenchmarkConfig` includes all provider related information which are necessary for deployment. Currently `SeMoDe` is limited to AWS, but an extension for other providers is work in progress. As for the calibration case, the provider related information are embedded for sake of simplicity.

When conducting a benchmark experiment and invoking endpoints, the execution data is stored in the corresponding logging service of the platform. In case of AWS, this service is CloudWatch[27]. When we fetch the data from this service, we analyze it and store the execution metrics in our `performance_data` table. Furthermore, we locally log the start and end time of each request as well as a unique identifier (UUID). Those pieces of information, together with the result of the platform (returned HTTP body from the API invocation) are stored in the tables `localrestevent` and `provider_event`. Based on the unique identifier (`platform_id` in `provider_event`), the tool is capable of matching the local REST event and the provider execution data from the specific log handler and associate the provider events with the corresponding performance data. In cases where, for example, the API gateway timeout is reached or unforeseen errors occur, no local rest events respectively provider events exists which limits the possibilities to investigate the client perceived overhead. However, the data for investigating the platform can be stored in either way.

The database connection is configured via properties. For changing the default value which assumes a running PostgreSQL at `http://localhost:5432` with a `postgres` user secured by `admin` password, all database related properties have to be altered.

4.2 Package Diagram

Under `src/main/resources` Spring Boot's `application.properties` can be found where the settings for all properties, e.g. database connection, logging levels, custom environment variables etc. can be changed. The Thymeleaf HTML templates are stored in the `templates` folder in the same directory together with the `static` content like css, images and javascript code.

The application code is located under `src/main/java` (gradle convention) and the classes are placed in a package structure which will be explained in the following. This report only describes the structure and some classes which are especially important for the architecture of the prototype and highlights design decisions and extension points. The main package `de.uniba.dsg.serverless` contains the classes for starting the spring boot application. Since I implemented a Spring web application, a web server (Tomcat) is started by default. To suppress this and use our prototype as CLI tool, an `ArgumentProcessor` class is implemented which needs another system property set at runtime. See the comment within the argument processor class for further information on how to avoid starting the web server.

All other classes are included in four packages `simulation.load`, `users`, `cli` and `pipeline`. The first subpackage `simulation.load` contains the implementation of a simulation. When given an arbitrary load pattern and a simulation input (average execution time, cold start time and container warm period), the tool computes the number of cold starts and running instances at a given point [10]. The `users` subpackage contains the model, controller and services for the user management of the application. Furthermore, it contains the security configuration for the web controllers which is session-based via a custom login page and a prior registration in the web frontend. The REST API on the other hand, is secured on a request basis where a user has to authenticate upfront and receives a JSON Web Token (JWT). The `cli` package contains a `UtitlityFactory` class where a builder pattern is implemented. Via the name of the implemented `CustomUtility`, the specific function is

[27] `https://aws.amazon.com/de/cloudwatch/`

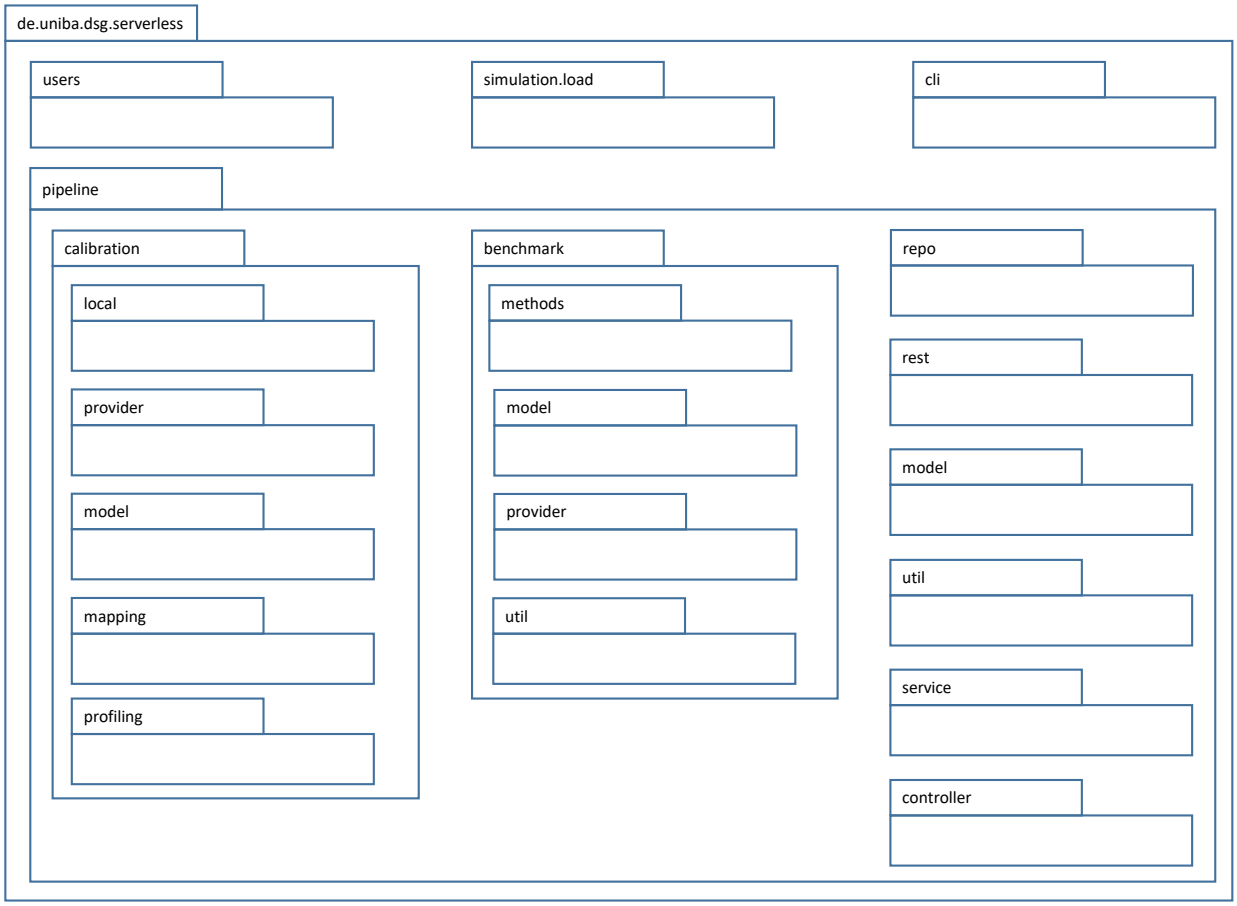


Figure 5: Package Diagram of `SeMoDe`

executed. This is also a mechanism to allow for an easy extension of the CLI features and test some functionality before integrating it into the pipeline as well as to provide stand alone features.

The most important package is `pipeline`. It contains the business logic for the benchmark and simulation pipeline. Before talking about these two packages, I will describe the other subpackages. All JPA related interfaces are included in `repo`. Spring Data JPA is used as on Object Relational Mapping (ORM) implementation. It generates the queries at compile time based on a declarative, human readable approach[28]. Where this naming scheme is insufficient for expressing queries, we use native queries and interface based Data Transfer Object (DTO) projection[29]. In our `model` package, all configuration related classes are included, see Figure 4. The `benchmark.model` and `calibration.model` classes contain the specific business related model classes like `performance_data` or `calibration_event` known from Figure 4. `controller` and `service` contain classes which are annotated with Spring stereotype of the same name. The architectural decision is to have no direct access from a controller to a repository class, so a controller class has only dependency injected service objects, but no repository objects. Furthermore, the service classes do not contain any state since the only source of truth is the database. Therefore, scaling and concurrency issues introduced by the middleware do not occur in `SeMoDe`. The `util` package includes the

[28] `https://docs.spring.io/spring-data/jpa/docs/current/reference/html/#jpa.query-methods.query-creation`
[29] `https://www.baeldung.com/spring-data-jpa-projections#interface-based-projections`

custom `SeMoDeException` as well as some static helper classes and Spring beans. As already mentioned in the introduction to this Section, we also expose a REST API implemented in the `rest` package. This package also contains subpackages for `controllers`, `dtos`, `services` and a `security` implementation via JWT.

The missing two packages `calibration` and `benchmark` implement the centerpiece of `SeMoDe`. The prototype contains a few generic interfaces which are extension points to enable the support for different FaaS platforms and establish a generic interface to deal with their heterogeneity. Two of them are `BenchmarkMethods` and `LogHandler` in the `benchmark` package. The first is included in the `methods` subpackage, where currently only AWS is supported. This interface defines methods for deploying, starting, fetching data and undeploying cloud functions to keep classes like the `BenchmarkExecutor` vendor independent. The latter is included in `provider` package where the major four cloud function platforms (AWS, Azure, Google and IBM) are supported for investigating the logs and enable a vendor specific casting of the log data to generate `performance_data` entries.

The third and last extension point is the interface `CalibrationMethods` in `calibration.provider`. This mechanism is similar to the benchmark extension at `BenchmarkMethods`. Here, `AWSCalibartion` and `LocalCalibration` implement this interface to deploy, start and undeploy the calibration. A local deployment is of course not necessary since we execute a Docker container to get the `calibration_events`. The other subpackages deal with different steps in the calibration and simulation pipeline. `local` contains the classes for the local calibration of a machine's hardware based on the execution of Docker containers and a calibration method. `mapping` contains the statistical evaluation of the tool with linear regression models, computation of the coefficient of determination (R^2) and a `MappingMaster` to compare different environments. `profiling` as the last subpackage of `calibration`, includes the classes for the local execution and simulation of a function based on the mapping information from the computed linear models.

5 Setup

5.1 System Setup

Since the prototype is implemented in Java, we need an installed Java Runtime Environment (JRE) version 11 or higher. Furthermore, to run the PostgreSQL database and use the calibration feature, we have to install Docker[30] since our local simulation approach will highly depend on Docker. Since we use the Java API client for Docker[31], a user has to set the host and port, `DOCKER_HOST` system property, and the `DOCKER_TLS_VERIFY` property to false. This is only relevant for users under Windows since the system defaults support is tied to Linux. The last setup step is to install `npm` and run `npm install` in `src/main/resources/static/js`.

[30] `https://www.docker.com`
[31] `https://github.com/docker-java/docker-java`

5.2 Provider Setup

Since only AWS is supported at time of writing, I will explain the steps which are needed to get the interaction with AWS work in the following.

1. Install the AWS CLI and configure it with a public and private key[32]. This is a prerequisite to ensure the prototype has access to the AWS services and can deploy cloud functions, invoke functions via gateways and read logs. The best way to do this is to create another user in the AWS Identity and Access Management (IAM) console and use this user and its credentials for the prototype invocations. It needs at least access to S3, ApiGateway, Lambda and CloudWatch.
2. You have to create a new IAM role for your executing lambda functions to specify the permissions, which the Lambda functions need. The following AWS managed policies are attached to this execution role:
   - AWSLambdaFullAccess
   - AWSCodeDeployRoleForLambda
   - AWSLambdaExecute
   - AWSLambdaBasicExecutionRole

5.3 System Startup

I specified a `docker-compose` file for the database container. To start up the container, run the `docker-compose up` command within the root directory of the application. Do not start `SeMoDe` within a container since we use the running Docker environment in the course of the simulation pipeline and need full access to the system. Due to the shared database[33], a configuration of our prototype respectively the experiments on one machine with a web browser and a second machine (preferably a Linux server) for running the experiments is desirable. Without other custom applications running on this Linux server, results can be derived as clean as possible.

To start the application, run the `gradlew bootRun` command. This automatically downloads the gradle distribution used and executes the wrapper to start the application.

6 Supported Functionality

This section explains the supported functionality and highlights some important design ideas, classes, code and publications where `SeMoDe`'s capabilities contribute to the technical realization of the presented ideas.

[32] `https://docs.aws.amazon.com/cli/latest/userguide/cli-configure-quickstart.html`
[33] Change the `application.properties` and especially the `spring.jpa.datasource.*` properties when you want to integrate over the database.

6.1 Web Interface

6.1.1 Pipeline Configuration

One important functionality is the pipeline configuration. Since I implemented a Spring Boot Web application, a configuration is possible via the web browser. As already mentioned, I use Thymeleaf as a HTML templating engine. In the following we will see different screenshots of this web interface together with a short explanation of the various input fields.

Setup Configuration

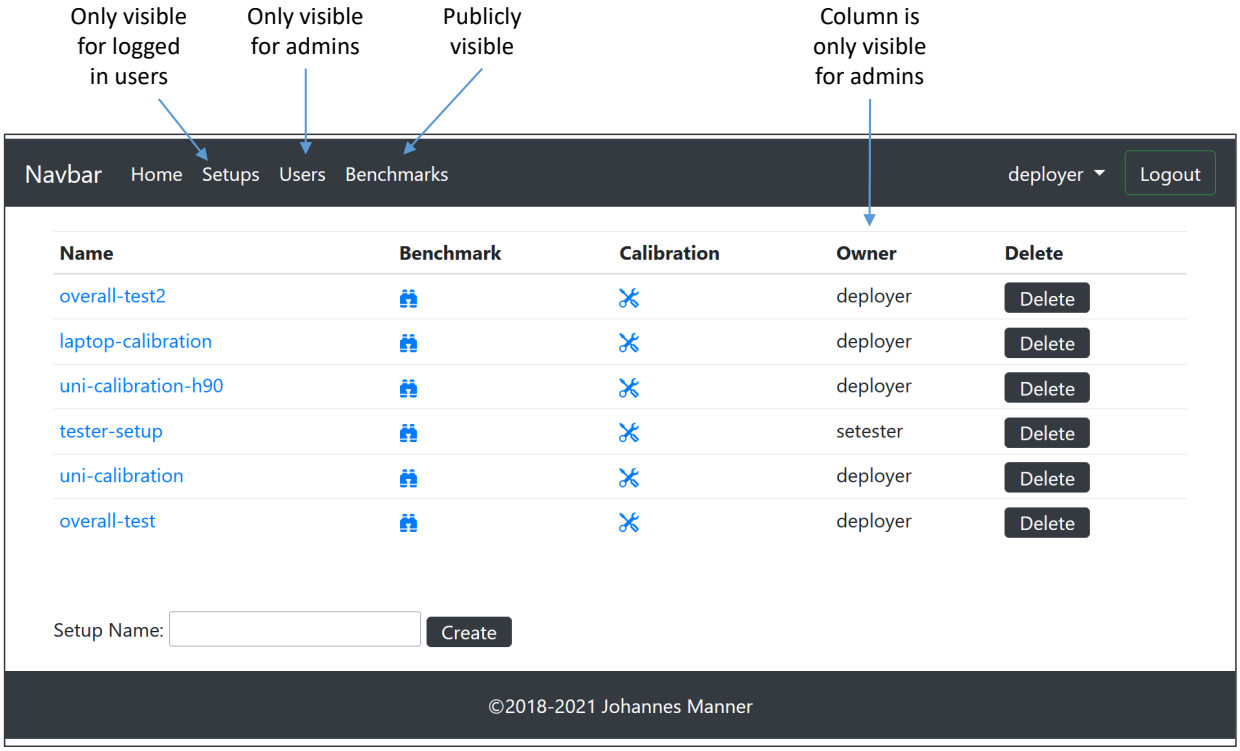


Figure 6: Setups Page to See Existing Setups and Create Them

There is a dualism in the way the pipeline setup will be configured. This is also visible in Figure 4 where two configuration classes dominate the database layout or Figure 5 which shows that business logic is mainly implemented in the two packages benchmark and calibration. Figure 6 depicts the main setup page, where new configurations can be created. The screenshot shows the application when a user is logged in. We use Spring Security and especially the Thymeleaf security integration to decide - based on the role - which elements of the page will be rendered. In this case, the logged in user *deployer* has also the administrator role and can therefore see the *Users* hyperlink[34] in the navbar and the *Owner* column. Via the binoculars, a user can go to the benchmark configuration page and via the tools icon to the calibration detail page.

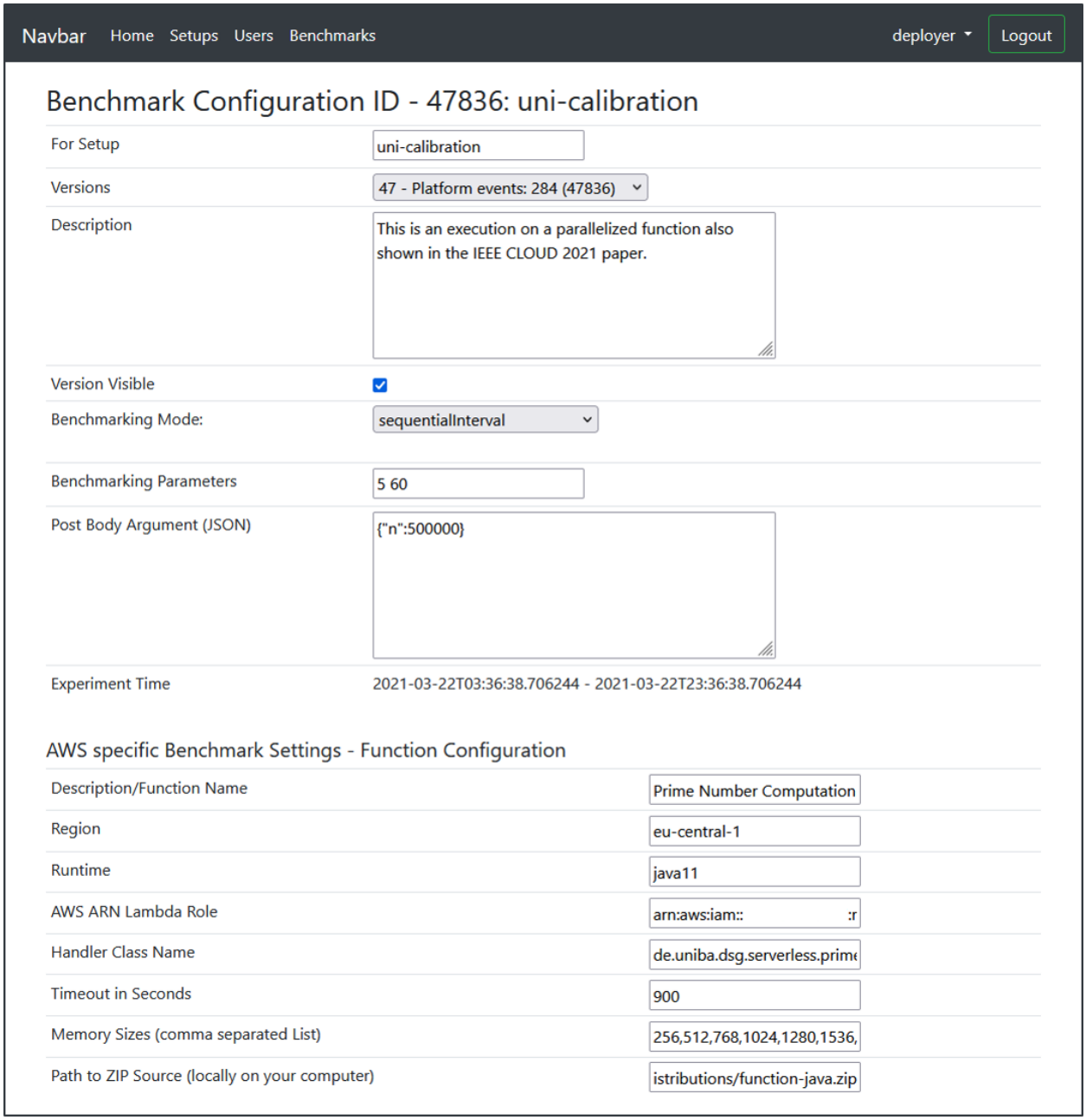


Figure 7: Benchmark Configuration Page I

Benchmark Configuration

Figures 7 and 8 show the screenshots for the `setups/{setupName}/benchmark`[35] endpoint. The first screenshot contains all editable input fields to configure the benchmark. Via the *Versions* dropdown, a user can see the configuration of older versions. Technically, asynchronous requests via JavaScript (*AJAX*) are made to an endpoint to get the benchmark configuration based on the selected version. Only the latest version is editable for the input fields, despite the description and version visible fields, which are editable for every version. The dropdown information is structured as follows:

"{version} - Platform events: {perfromance data entries} (database id)"

The number of performance data entries in the database indicate if a benchmark was executed based on this version and if the data was fetched from the corresponding provider. The performance entries and especially the `memory_used` and `precise_duration` are later used for displaying the data graphically as can be seen in Figure 8. The primary key of the benchmark configuration is also displayed in parenthesis for an in-depth investigation and custom queries when accessing the database as an administrator.

A custom *Description* is the next input field. This can be used to make notes about the configuration during the experiment planning and execution, but also for publicly marked experiments when the field *Version Visible* is set. Then a non-authenticated user can see a subset of the information presented here, more details about the publicly available website will be given in a subsequent Section.

The following three fields are connected to each other. Different *Benchmarking Mode*s are implemented to support clean test beds as well as arbitrary load patterns. Further details on the benchmarking modes implemented and the load pattern generation can be found in Section 6.2.2. Based on the selected mode, different *Benchmarking Parameters* have to be specified to define the invocation process. Since the prototype is currently only capable of invoking REST endpoints, we added some *Post Body Arguments* in JSON format as input for the cloud function. When a benchmark is executed, the experiment start and end time is set, displayed at *Experiment Time* and overwritten with null values when the next version is created and stored in database.

As mentioned before, currently only AWS is supported. Therefore, in the next paragraph of the website, the AWS specific function configuration is set. The first fields are self-explanatory, where the *Description/Function Name* is added to the overall description. This enables the user to describe the function in this context. The *AWS Amazon Resource Name (ARN) Lambda Role* is the identifier of the created role presented in Section 5.2. All other fields are similar to the fields at the AWS dashboard where you can also use the ZIP upload option.

Deployment Internals are the next paragraph and completely filled and cleared by the tool, see Figure 8. This procedure enables a check and possible intervention of the experimenter when some components malfunction. It is furthermore needed to undeploy the deployed function at a later point in time. Via the *Update Configuration* button, a user of the tool can store a new version of the configuration. It is disabled in the screenshot since version 47

[34] An explanation of the user's page is done in Appendix A.

[35] I use a curly bracket semantics in the style of specifying controller paths in Spring Web for dynamically changing fields.

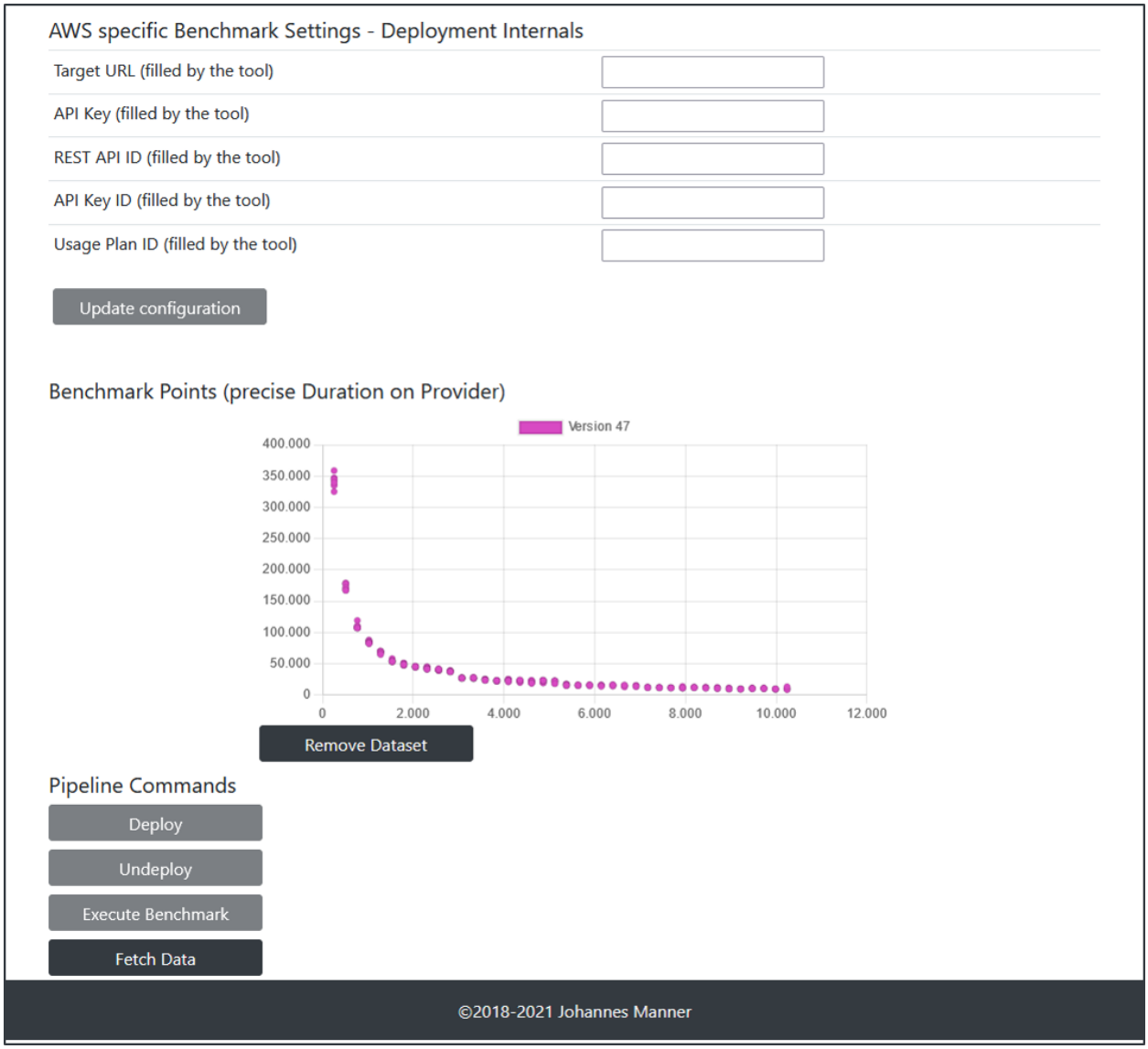


Figure 8: Benchmark Configuration Page II

is not the most recent version, but a version with *Benchmark Points* as can be seen by the diagram. Whenever a version with associated performance data is selected, a new set of data points is included in the diagram where the last version data can also be removed via the *Remove Dataset* button. Also the actions *Deploy*, *Undeploy* and *Execute Benchmark* are disabled. *Fetch Data* from a provider's logging service like AWS CloudWatch is possible when an experimenter forgot to do so, but already fetched data cannot be fetched again.

Calibration Configuration

Figures 9, 10 and 11 show screenshots for the `setups/{setupName}/calibration` endpoint. The structure of the calibration webpage is similar to the benchmark webpage described in the previous Section. Due to this similarity, I will only describe new fields and elements. *Local Configuration* section includes the configuration for the local experiment machine and the upfront calibration of the used hardware. The *Local Steps* property describes the resource increase for the start of a Docker container, whose location is specified in the *Docker Source Folder* property. This CPU resource limitation is done via the `cpus` command line argument when starting a container. The *Number of local calibrations* describes how often the calibration is executed on every *Local Steps* setting. A result of this calibration can be seen in Figure 10 in the first diagram.

In the AWS configuration, the only change is the *Bucket name* where the output of the calibration function should be stored. The mechanism here is different to the benchmarking example since the API Gateway runs into a timeout after 30 seconds and cannot get the response directly from the platform. Additionally, the `LogHandler` for AWS does not parse the execution data of the calibration function. Therefore, the result of the calibration on AWS Lambda is fetched from this bucket and analyzed via the `LinpackParser` class. Linpack [12, 13] is currently the only option to calibrate the hardware stacks and sample function are included in the `functions` folder of our prototype[36].

The next paragraph in the webpage is a mapping step from the local calibration (and therefore the local machine stack) and the provider calibration. Based on the two calibrations, the tool computes two linear regressions for every architectural stack. It equals the two equations and computes local CPU equivalents for defined platform configuration as seen in the *Memory Sizes on Provider Platform* field. It is possible due to the two dropdown fields to select different configurations, which is especially interesting when changing BIOS or kernel settings to influence the CPU frequency scaling [2], the algorithm used to balance performance and power consumption or turbo boost options to name only a few. Currently, selected calibrations are displayed in brackets and their data is used for generating the two diagrams. The *Machine Configuration* paragraph is for documentation purposes to describe fair and repeatable benchmarks with as much information as possible.

In Figure 11, a user describes the *Folder* where the source code of the function under test, which is a candidate for a cloud function deployment, is located. *Environment Variables* can be passed by a file to `SeMoDe` which sets the variables when starting the containers. The *Number of Profiles* for every memory size configured in the previous screenshot is set. Via the dropdown *Previous Profiles*, a user can select a profile and sees also the version number of the calibration. This enables a user to select the version dropdown in Figure 9 to see the configuration yielded to this profile. The buttons are similar to the previous benchmarking

[36] `https://github.com/johannes-manner/SeMoDe/tree/master/templates`

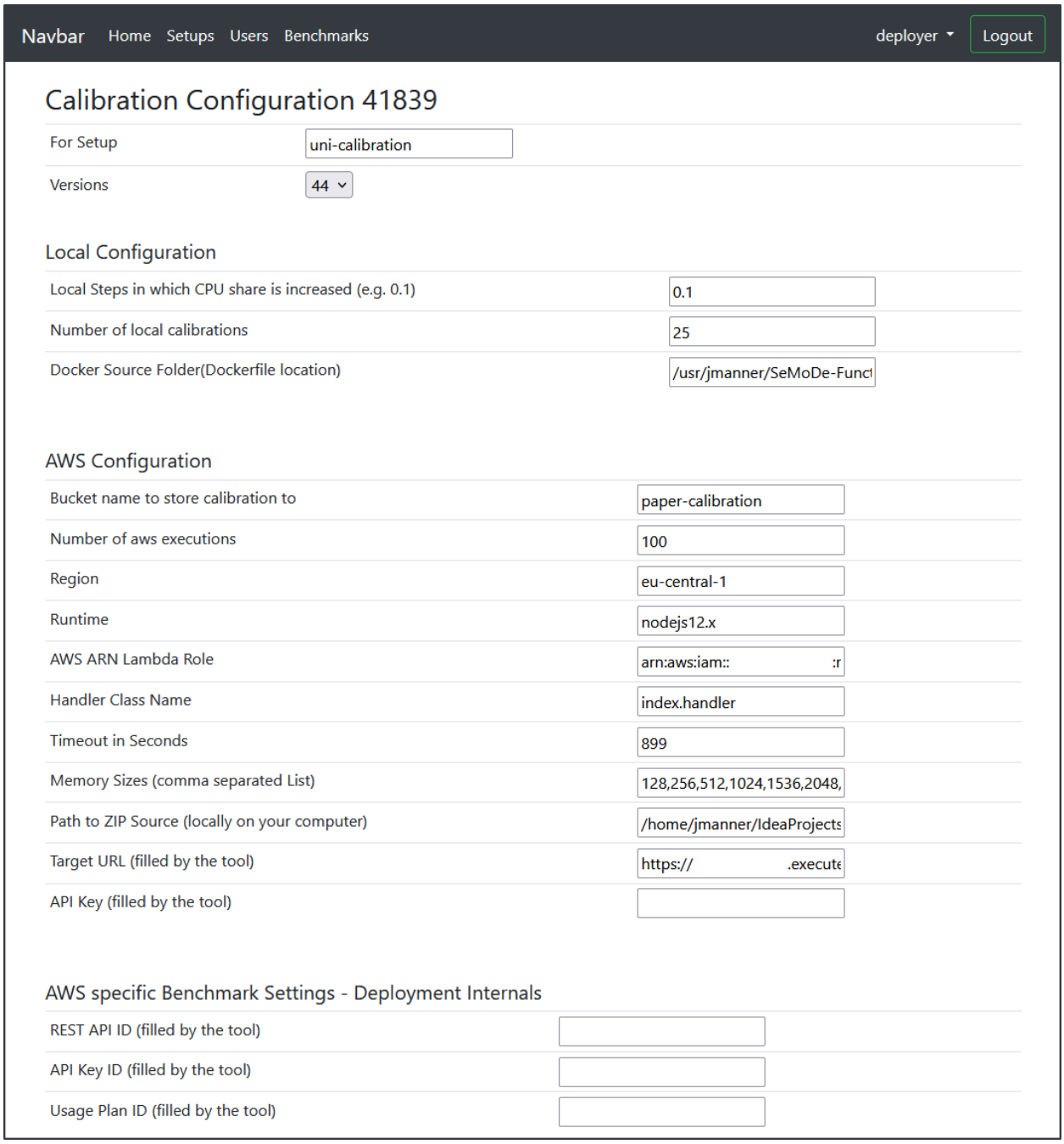


Figure 9: Calibration Configuration Page I

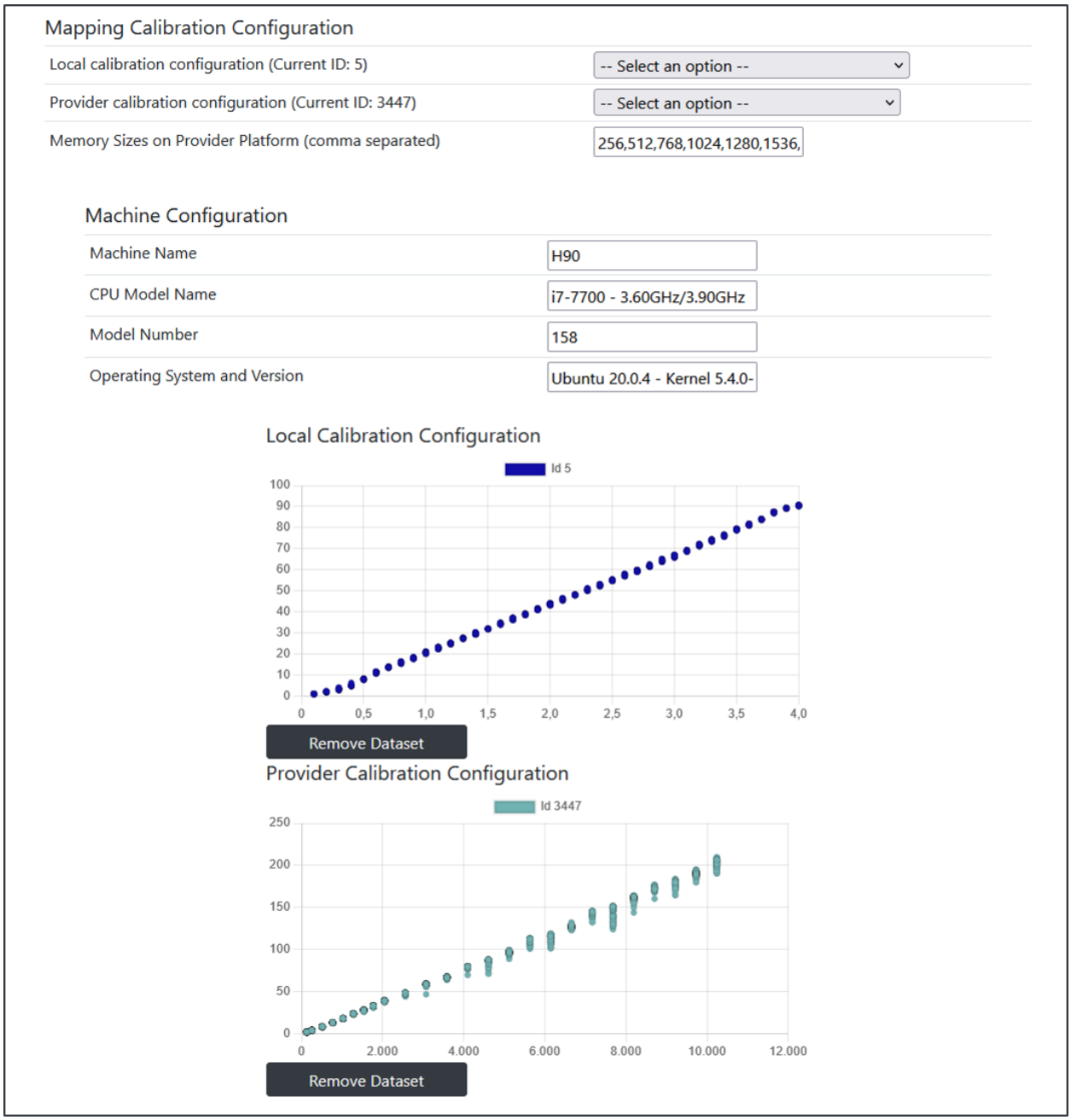


Figure 10: Calibration Configuration Page II

Figure 11: Calibration Configuration Page III

case. *Deploy Calibration*, *Start Calibration* and *Undeploy Calibration* are referred to AWS. All other buttons should be self explanatory via their labeling.

6.1.2 Visualizing Execution Data

In Figure 7, there was a field which marks the configuration and the benchmarking results as publicly available. Figure 12 shows the condensed benchmarking information and the possibility to download the data (via the Download button) for further analysis.

6.2 Benchmarking Cloud Function Platforms

6.2.1 Implementing a Function under Benchmark

First discussed in the database layout, the function under benchmark is invoked via synchronous HTTP calls. To assess the user perceived performance, the start and end time on the user side are logged as depicted in Figure 13. This is only possible when the API gateway does not timeout. In case of AWS, the timeout is about 30 seconds. If the request times out, the invocation succeeds and the start time is logged but the end time is missing as well as the `provider_event`. In this case only the `performance_data` information is present when fetching the data. This is important to have in mind when designing experiments.

In Figure 13, four invocations are presented, where the orange part is the duration perceived on the user side and the blue period is the execution time on the platform. This enables an assessment of round trip times and routing/processing on the FaaS platform. We implemented a custom response to enable a mapping of the local and provider data which will be discussed later on. This custom response includes all fields of the `provider_event` table or `ProviderEvent` class except for the id field. Especially the CPU information is important to deal with the heterogeneity of hardware at different regions of a FaaS provider as other research has shown performance variations, see [14, 15, 16].

6.2.2 Benchmarking Modes

As already shown in Figure 7, there is a dropdown for selecting the benchmarking mode. The tool has five benchmark modes. The first four modes are artificially created to draw strong conclusions when conducting experiments. The last option is for real world traces as described in the following enumeration. For each mode, the mandatory attributes are described and can be configured in the *Benchmarking Parameters* input field, see also Figure 7:

- **Mode `concurrent`** - Invoking the function under test once in parallel by executing N requests.
    1. Number of requests.
- **Mode `sequentialInterval`** - Sequentially triggering a function with a fixed time interval between the starting point of the corresponding function invocations. This

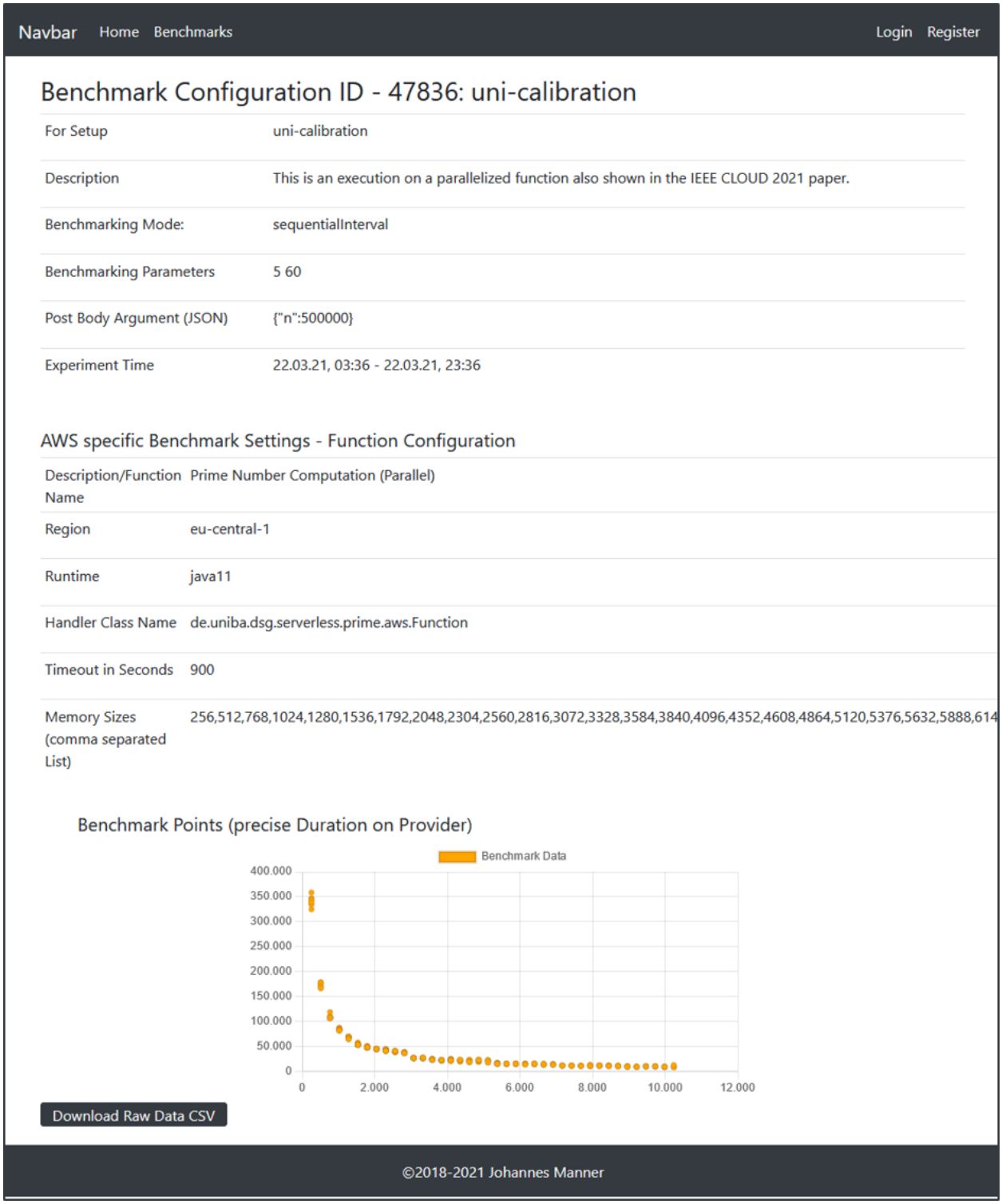


Figure 12: Benchmark Publicly Visible Page

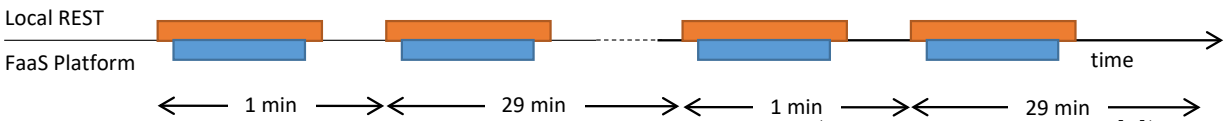


Figure 13: User and Platform Perceived Performance (based on Figure 1 in [1])

can lead to situations, where the invocations are not sequential because the execution time is greater than the interval. In such a case, the platform instantiates further cloud functions to serve the requests.

1. Number of requests.
2. Seconds between the request execution start times.

- **Mode `sequentialConcurrent`** - Combines the previous two invocations modes.
  1. Number of executions groups.
  2. Number of requests in each group.
  3. Delay between termination of group g and start of group g+1 in seconds.
- **Mode `sequentialChangingInterval`** - This mode triggers functions in an interval with varying delays between execution start times in a round robin fashion.
  1. Number of requests.
  2. List of delays.
- **Mode `arbitraryLoadPattern`** - The function endpoints are triggered based on a csv file. The file contains only one column and for each row a double value with a relative timestamp compared to the actual time when the generation is started, e.g. 0 means now and 5 means now + 5 seconds. This value indicates when a specific REST call, invoking the cloud function at the platform, should be submitted.
  1. File path of the csv load pattern file.

`SeMoDe` generates a csv file for the first four modes via the `LoadPatternGenerator` class. This file is located in the folder `setups/setupName/benchmark/loadPattern.csv` and overwritten when further benchmarks are executed based on the same configuration.

6.2.3 Submitting Requests based on Arbitrary Load Pattern

Listing 1 contains the centerpiece of the benchmark invocation process. Since the prototype generically implements the `BenchmarkMethods` interface, I pass a list of deployed methods with their endpoints to this function. For the invocation, I implemented two executor services which handle the concurrent execution of tasks via thread pools. The reason for these two executors is, that we want to schedule requests based on our generated load pattern. This is only possible with a `ScheduledExecutorService`[37], but this executor service is limited in the number of concurrent threads which are determined by the factory method's `corePoolSize` attribute. Since I do not know the concurrency level a priori due to the arbitrary load

[37] `https://docs.oracle.com/en/java/javase/11/docs/api/java.base/java/util/concurrent/ScheduledExecutorService.html`

pattern and function execution time, the core pool size cannot be determined. The problem is, that there is no runtime error when the scheduled requests exceed the core pool size, but the requests are then queued by the work queue of the thread pool and not executed on the time specified by the load pattern since `SeMoDe` requests the cloud function in a synchronous way and therefore the executing thread blocks (IO wait).

Listing 1: Centerpiece of the `BenchmarkExecutor` Class

```
1  public List<LocalRESTEvent> executeBenchmark(final List<BenchmarkMethods>
       benchmarkMethodsFromConfig) {
2
3    final List<Double> timestamps = this.loadLoadPatternFromFile();
4
5    final ScheduledExecutorService executor = Executors.newScheduledThreadPool(processors());
6    final ExecutorService delegator = Executors.newCachedThreadPool();
7
8    final List<Future<LocalRESTEvent>> responses = new CopyOnWriteArrayList<>();
9
10   long tmpTimestamp = 0;
11   // for each provider
12   for (final BenchmarkMethods benchmarkMethods : benchmarkMethodsFromConfig) {
13     if (benchmarkMethods.isInitialized()) {
14       // for each function
15       for (final String functionEndpoint : benchmarkMethods.getUrlEndpointsOnPlatform()) {
16         // for each timestamp
17         for (final double timestamp : timestamps) {
18           tmpTimestamp = (long) (timestamp * 1000);
19           final FunctionTrigger f = new FunctionTrigger(benchmarkMethods.getPlatform(),
                 benchmarkConfig.getPostArgument(), new URL(functionEndpoint));
20           FunctionTriggerWrapper fWrapper = new FunctionTriggerWrapper(delegator, responses, f);
21           executor.schedule(fWrapper, tmpTimestamp, TimeUnit.MILLISECONDS);
22         }
23       }
24     }
25   }
26
27   // Shut down the first scheduled service. This means that all wrapper function trigger tasks are
28   // run and the function trigger tasks are submitted.
29   // Time to wait is the last timestamp from now on executing a function.
30   this.shutdownExecService(executor, tmpTimestamp + 30_000);
31   // Wait for the function trigger tasks to terminate (300 seconds timeout)
32   this.shutdownExecService(delegator, tmpTimestamp + 300_000);
33
34   // collecting the custom responses from the cloud platform
35   List<LocalRESTEvent> events = new ArrayList<>();
36   for (Future<LocalRESTEvent> futureEvent : responses) {
37     events.add(futureEvent.get());
38   }
39   return events;
40 }
```

This is the reason why `executor` is only used for asynchronously starting the requests which is a millisecond task. The synchronous execution is done by a `CachedThreadPool`[38] named `delegator` in the snippet which dynamically scales the number of threads in its thread pool. In line 19, the function trigger, a callable which synchronously executes the HTTP request, is created. This callable is wrapped in a runnable in line 20, which also gets a reference to the delegator (the autoscaling executor service) as well as a reference to the thread-safe list where the `Futures`[39] (handle to get the result of the computation later on) are collected. In line 21,

[38] `https://docs.oracle.com/en/java/javase/11/docs/api/java.base/java/util/concurrent/Executors.html#newCachedThreadPool()`

[39] `https://docs.oracle.com/en/java/javase/11/docs/api/java.base/java/util/concurrent/`

the executor schedules the wrapped function triggers. Dependent on the start time of the scheduled callable, it executes a call to the delegator which executes the `FunctionTrigger`. After submitting the last request, the scheduled executor service terminates immediately in line 30. The call in line 32 blocks until the last synchronous invocation returns. It waits the timeout period for the cloud function to be sure that all functions can terminate correctly on the selected platform. After the execution, the futures and especially the results are added to the `events` list in 37. In the following they are stored in the database.

`SeMoDe` is dependent on the system resources and the number of threads a system can handle. Another limiting factor is the amount of memory the system is capable of for the number of concurrent functions which can be executed in parallel. In this aspect, the implementation is similar to other load testing tools like JMeter[40].

6.3 Simulating Local Cloud Function Executions

6.3.1 Calibrating Hardware

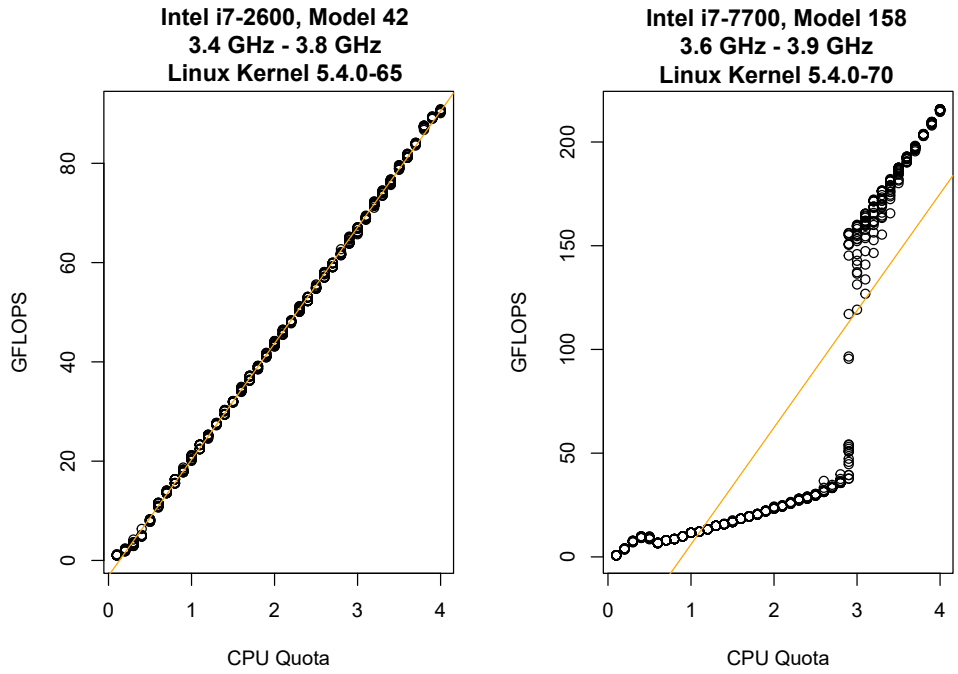


Figure 14: CPU performance computed by LINPACK at different CPU Quotas (based on Figure 1 in [2])

During the research on a simulation and benchmarking pipeline for cloud functions, I discovered an interesting non-linear scaling of CPU performance [2]. An execution of LINPACK showed the behavior as shown in Figure 14. GFLOPS are computed upon the LINPACK result which is parsed by the `LinpackParser`. Detailed data is stored in the folder `/setups/-{setupName}/calibration`. At this location, the output of the LINPACK computation is stored, too. As already said, the tool artificially increase the CPU share/quota by using Docker's cgroup mechanism via the `cpus` parameter. Lilja [17] described *Linearity* as a

`Future.html`

[40] `https://jmeter.apache.org/`

characteristic of a good performance metric. To achieve this linearity and detect cases as in the right side of Figure 14, I implemented a `Hardware Calibration Feature`[41]. This feature computes a linear regression and the R^2 which shows whether the performance scales linearly on the corresponding local machine. If this is not the case, as for the i7-7700 machine, a user has to adjust the kernel and/or bios settings of the machine to enable a linear scaling of CPU resources which can be checked again with the calibration feature. Otherwise the calibration data cannot be used for the simulation approach since the performance is unpredictable for some utilization levels, e.g. for 2.9 to 3.4 CPU quotas at i7-7700.

If the calibration is executed as part of an experiment, a user can also see the data points as shown in Figure 10 (second diagram). The tool includes two properties as described in the README where a user of the tool can specify the number of runs for each CPU quota and the steps in which the CPU quota is incremented. The default values are 1 run (`runs`) with steps of 0.1 (`steps`). A single run is sufficient to get a confirmation through the statistical output if the compute performance scaling is appropriate for conducting the simulation experiments.

6.3.2 Simulation based on Calibration

The calibration information is used to formulate an equation to compute the CPU quota on a local machine based on a potentially interesting memory setting on the cloud provider to predict the execution behavior of the function under test. The results computed there are now used in our `ContainerExecutor` class when starting the containers with the correct CPU quota. As a result of the container execution, the information are parsed, saved in files at `/setups/{setupName}/calibration/profiles` and stored as `ProfileRecord`s in the database. These records are then displayed at the web interface, see Figure 11, and give a first hint whether the cloud function would profit proportionally from an resource increase on the platform. The interested reader is referenced to our paper on *Optimizing cloud function configuration via local simulations* [9] where a statistical evaluation of the described local simulation and prediction is made. This paper also gives the answer to our first statement **S1** where a novel dev-prod parity concept is introduced.

6.4 REST API

6.4.1 REST API Security

As mentioned in Section 4.2, the basic security configuration is session based, see `SecurityConfig.WebSecurityAdapter` class. For the REST API, I included another `Configuration` class (`SecurityConfig.ApiSecurityAdapter`) to handle requests and secure them via JWTs[42]. Therefore, I implemented a custom filter class (`JwtAuthenticationTokenFilter`) and registered it in the pipe and filter security architecture of Spring Boot Security. A user has to send an `AuthenticationRequest` to the endpoint `/api/login` with the username and password. The tool generates a `JWTTokenResponse` with a Bearer token. In the following, the user has to include this token in every request as an authorization header. After some

[41] `https://github.com/johannes-manner/SeMoDe#hardware-calibration-feature`
[42] `https://datatracker.ietf.org/doc/html/rfc7519`

time (specified via the `jwt.expiration` property), the token expires and the user has to resend the authentication to get a new token.

6.4.2 Exposed REST API Endpoints

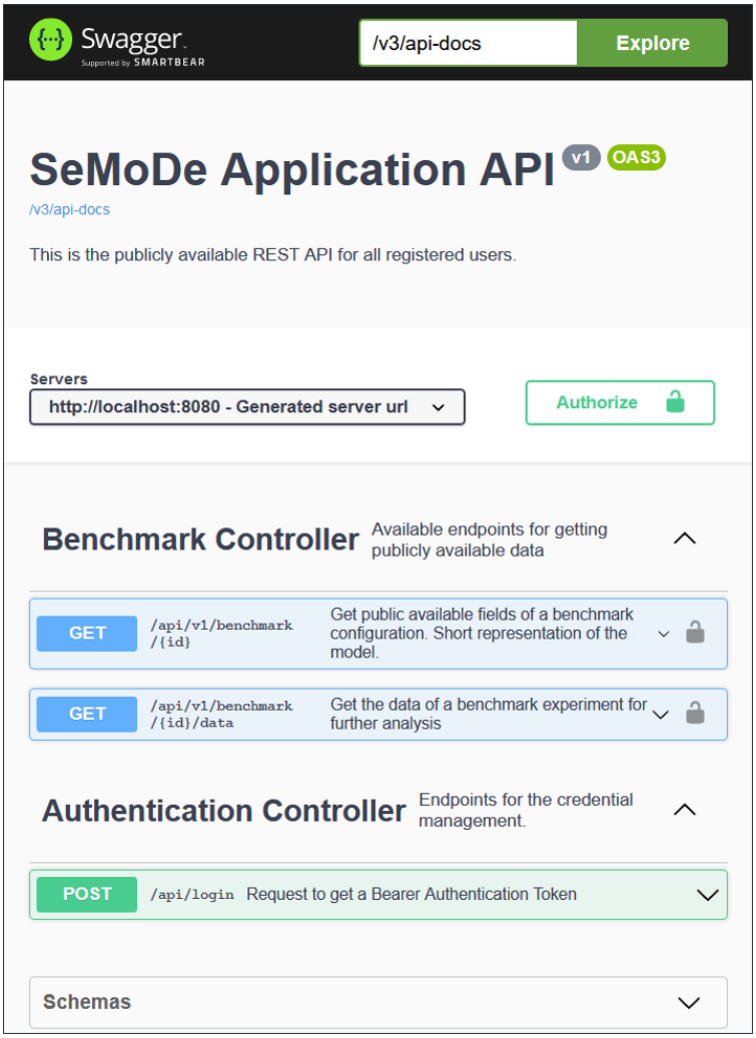


Figure 15: OAS for exposed REST API.

Figure 15 shows the Swagger UI displaying the OAS of our prototype. It highlights, indicated by the lock, the authorization needed as described in the previous Section. At the `/swagger` endpoint, the interested reader can access OAS and the benchmark configurations which are publicly available via the `/benchmarks` endpoint. The REST API is an additional option to access the data and at an early stage.

6.5 CLI Features

Since parts of the overall benchmarking and simulation pipeline consume hours or even days to complete, I implemented a CLI feature to run the application on the corresponding Linux machines and to avoid using the web browser since the experiment machines are normally based on a server image without a graphical user interface.

The command in Listing 2 shows the invocation of the java application. It is vital to set the property `spring.main.web-application-type` to `NONE` here to suppress Spring's default behavior to start a web server (TomCat). The ampersand detaches the process from the console and via `> out.txt` the standard output is written to the file `out.txt`. This enables an error handling and investigation of details when running the prototype.

Listing 2: Start `SeMoDe` as CLI application

```
1 $ java -jar -Dspring.main.web-application-type=NONE app.jar & > out.txt
```

7 Conclusion and Future Work

In this technical report, I documented `SeMoDe`, a research prototype for benchmarking cloud functions and simulating their local behavior to predict the runtime behavior of function in the cloud locally. Furthermore, I performed an SLR which gives insights in the current state of FaaS benchmarking and simulation approaches and led to a condensed checklist to improve the quality of the documentation and insights of research in this field.

Since the current implementation is only tackling a single function in isolation by conducting microbenchmarks which is often a main factor of criticism, I plan to extend the scope of the simulation component to small architectures comprising typical interacting services like databases, gateways and message queues. A first idea is to use docker-compose[43] for orchestrating the solution and executing the different services. Another aspect for future work is to extend the publicly available experiments and publish them on our prototypes website[44] to support **S3**.

[43] `https://docs.docker.com/compose/`
[44] `https://semode.pi.uni-bamberg.de`

References

[1] J. Manner, M. Endreß, T. Heckel, and G. Wirtz, "Cold Start Influencing Factors in Function as a Service," in *Proceedings of WoSC*, 2018.

[2] J. Manner and G. Wirtz, "Why many benchmarks might be compromised," in *Proceedings of SOSE*, 2021.

[3] E. van Eyk, L. Toader, S. Talluri, L. Versluis, A. Uta, and A. Iosup, "Serverless is More: From PaaS to Present Cloud Computing," *IEEE Internet Computing*, vol. 22, no. 5, pp. 8–17, Sept. 2018.

[4] A. Slominski, V. Muthusamy, and V. Isahagian, "The Future of Computing is Boring (and that is exciting!)," in *Proceedings of (IC2E)*, 2019.

[5] P. Mell and T. Grance, "The NIST definition of cloud computing," National Institute of Standards and Technology, Gaithersburg, Tech. Rep., 2011.

[6] J. Kuhlenkamp and S. Werner, "Benchmarking FaaS Platforms: Call for Community Participation," in *Proceedings of WoSC*, 2018.

[7] K. Huppler, "The Art of Building a Good Benchmark," in *Proceedings of TPCTC*, 2009.

[8] S. Kounev, K.-D. Lange, and J. von Kistowski, *Systems Benchmarking*. Springer International Publishing, 2020.

[9] J. Manner, M. Endreß, S. Böhm, and G. Wirtz, "Optimizing cloud function configuration via local simulations," in *Proceedings of IEEE CLOUD*, 2021.

[10] J. Manner and G. Wirtz, "Impact of Application Load in Function as a Service," in *Proc. of SummerSoC*, 2019.

[11] J. Manner, S. Kolb, and G. Wirtz, "Troubleshooting serverless functions: a combined monitoring and debugging approach," *SICS Software-Intensive Cyber-Physical Systems*, vol. 34, no. 2-3, pp. 99–104, 2019.

[12] J. J. Dongarra, C. B. Moler, J. R. Bunch, and G. W. Stewart, *LINPACK Users' Guide*. Society for Industrial and Applied Mathematics, 1979.

[13] J. J. Dongarra, P. Luszczek, and A. Petitet, "The LINPACK benchmark: past, present and future," *Concurrency and Computation: Practice and Experience*, vol. 15, no. 9, pp. 803–820, 2003.

[14] J. O'Loughlin and L. Gillam, "Performance evaluation for cost-efficient public infrastructure cloud use," in *Proc. of GECON*, 2014.

[15] I. Pelle, J. Czentye, J. Doka, and B. Sonkoly, "Towards latency sensitive cloud native applications: A performance study on AWS," in *Proceedings of CLOUD*, 2019.

[16] R. Cordingly, W. Shu, and W. J. Lloyd, "Predicting performance and cost of serverless computing functions with SAAF," in *Proceedings of DASC/PiCom/CBDCom/CyberSciTech*, 2020.

[17] D. J. Lilja, *Measuring Computer Performance*. Cambridge University Press, 2000.

[18] B. Kitchenham and S. Charters, "Guidelines for performing systematic literature reviews in software engineering," Keele University, Keele,UK and University of Durham, UK, Tech. Rep. EBSE-2007-01, 2007.

[19] V. Yussupov, U. Breitenbücher, F. Leymann, and M. Wurster, "A systematic mapping study on engineering function-as-a-service platforms and tools," in *Proceedings of UCC*, 2019.

[20] J. Scheuner and P. Leitner, "Function-as-a-service performance evaluation: A multivocal literature review," *Journal of Systems and Software*, vol. 170, p. 110708, dec 2020.

[21] E. V. Eyk, J. Scheuner, S. Eismann, C. L. Abad, and A. Iosup, "Beyond microbenchmarks: The SPEC-RG vision for a comprehensive serverless benchmark," in *Proceedings of ICPE*, 2020.

[22] S. K. Mohanty, G. Premsankar, and M. di Francesco, "An evaluation of open source serverless computing frameworks," in *Proceedings of CloudCom*, 2018.

[23] A. Das, S. Patterson, and M. Wittie, "EdgeBench: Benchmarking Edge Computing Platforms," in *Proceedings of WoSC*, 2018.

[24] H. Jeon, C. Cho, S. Shin, and S. Yoon, "A CloudSim-extension for simulating distributed functions-as-a-service," in *Proceedings of PDCAT*, 2019.

[25] K. Kritikos and P. Skrzypek, "Simulation-as-a-service with serverless computing," in *Proceedings of SERVICES*, 2019.

[26] D. Bardsley, L. Ryan, and J. Howard, "Serverless Performance and Optimization Strategies," in *Proceedings of SmartCloud)*, 2018.

[27] Y. Gan, Y. Zhang, D. Cheng, A. Shetty, P. Rathi, N. Katarki, A. Bruno, J. Hu, B. Ritchken, B. Jackson, K. Hu, M. Pancholi, Y. He, B. Clancy, C. Colen, F. Wen, C. Leung, S. Wang, L. Zaruvinsky, M. Espinosa, R. Lin, Z. Liu, J. Padilla, and C. Delimitrou, "An open-source benchmark suite for microservices and their hardware-software implications for cloud & edge systems," in *Proceedings of ASPLOS*, 2019.

[28] S. Malla and K. Christensen, "HPC in the cloud: Performance comparison of function as a service (FaaS) vs infrastructure as a service (IaaS)," *Internet Technology Letters*, vol. 3, no. 1, p. e137, dec 2019.

[29] M. Malawski, A. Gajek, A. Zima, B. Balis, and K. Figiela, "Serverless execution of scientific workflows: Experiments with hyperflow, AWS lambda and google cloud functions," *Future Generation Computer Systems*, vol. 110, pp. 502–514, 2020.

[30] V. Giménez-Alventosa, G. Moltó, and M. Caballer, "A framework and a performance assessment for serverless MapReduce on AWS Lambda," *Future Generation Computer Systems*, vol. 97, pp. 259–274, 2019.

[31] D. F. Quaresma, T. E. Pereira, and D. Fireman, "Validation of a simulation model for faas performance benchmarking using predictive validation," *CoRR*, vol. abs/2106.15555, 2021.

[32] A. U. Gias and G. Casale, "COCOA: cold start aware capacity planning for function-as-a-service platforms," in *Proceedings of MASCOTS*, 2020.

[33] G. McGrath and P. R. Brenner, "Serverless Computing: Design, Implementation, and Performance," in *Proceedings of ICDCSW*, 2017.

[34] D. Ustiugov, P. Petrov, M. Kogias, E. Bugnion, and B. Grot, "Benchmarking, analysis, and optimization of serverless function snapshots," in *Proceedings of ASPLOS*, 2021.

[35] R. N. Calheiros, R. Ranjan, A. Beloglazov, C. A. F. D. Rose, and R. Buyya, "CloudSim: a toolkit for modeling and simulation of cloud computing environments and evaluation of resource provisioning algorithms," *Software: Practice and Experience*, vol. 41, no. 1, pp. 23–50, 2010.

[36] T. Back and V. Andrikopoulos, "Using a microbenchmark to compare function as a service solutions," in *Proceedings of ESOCC*, 2018.

[37] R. Pellegrini, I. Ivkic, and M. Tauber, "Function-as-a-service benchmarking framework," in *Proceedings of CLOSER*, 2019.

[38] N. Somu, N. Daw, U. Bellur, and P. Kulkarni, "Panopticon: A comprehensive benchmarking tool for serverless applications," in *Proceedings of COMSNETS*, 2020.

[39] D. Jackson and G. Clynch, "An Investigation of the Impact of Language Runtime on the Performance and Cost of Serverless Functions," in *Proceedings of WoSC*, 2018.

[40] A. Kuntsevich, P. Nasirifard, and H.-A. Jacobsen, "A distributed analysis and benchmarking framework for apache OpenWhisk serverless platform," in *Proceedings of Middleware*, 2018.

[41] H. Lee, K. Satyam, and G. C. Fox, "Evaluation of Production Serverless Computing Environments," in *Proceedings of CLOUD*, 2018.

[42] H. Martins, F. Araújo, and P. R. da Cunha, "Benchmarking serverless computing platforms," *Journal of Grid Computing*, vol. 18, no. 4, pp. 691–709, 2020.

[43] J. Kim and K. Lee, "FunctionBench: A suite of workloads for serverless cloud function service," in *Proceedings of CLOUD*, 2019.

[44] ——, "Practical cloud workloads for serverless FaaS," in *Proceedings SoCC*, 2019.

[45] M. Copik, G. Kwasniewski, M. Besta, M. Podstawski, and T. Hoefler, "Sebs: A serverless benchmark suite for function-as-a-service computing," *CoRR*, vol. abs/2012.14132, 2020.

[46] M. Grambow, T. Pfandzelter, L. Burchard, C. Schubert, M. Zhao, and D. Bermbach, "Befaas: An application-centric benchmarking framework for faas platforms," *CoRR*, vol. abs/2102.12770, 2021.

[47] D. Bortolini and R. R. Obelheiro, "Investigating performance and cost in function-as-a-service platforms," in *Proceedings of 3PGCIC*, 2019.

[48] L. Wang, M. Li, Y. Zhang, T. Ristenpart, and M. Swift, "Peeking Behind the Curtains of Serverless Platforms," in *Proceedings of USENIX ATC*, 2018.

[49] W. Lloyd, S. Ramesh, S. Chinthalapati, L. Ly, and S. Pallickara, "Serverless Computing: An Investigation of Factors Influencing Microservice Performance," in *Proceedings of IC2E*, 2018.

[50] D. B. Pons and P. G. López, "Benchmarking parallelism in faas platforms," *CoRR*, vol. abs/2010.15032, 2020.

[51] J. Kuhlenkamp, S. Werner, M. C. Borges, D. Ernst, and D. Wenzel, "Benchmarking elasticity of faas platforms as a foundation for objective-driven design of serverless applications," in *Proceedings of SAC*, 2020.

[52] P. Maissen, P. Felber, P. G. Kropf, and V. Schiavoni, "Faasdom: a benchmark suite for serverless computing," in *Proceedings of DEBS*, 2020.

[53] M. Malawski, K. Figiela, A. Gajek, and A. Zima, "Benchmarking heterogeneous cloud functions," in *Proceedings of Euro-Par*, 2017.

[54] M. Pawlik, K. Figiela, and M. Malawski, "Performance evaluation of parallel cloud functions," in *Proceedings of ICPP (Poster)*, 2018.

[55] K. Figiela, A. Gajek, A. Zima, B. Obrok, and M. Malawski, "Performance evaluation of heterogeneous cloud functions," *Concurrency and Computation: Practice and Experience*, vol. 30, p. e4792, 2018.

[56] T. Yu, Q. Liu, D. Du, Y. Xia, B. Zang, Z. Lu, P. Yang, C. Qin, and H. Chen, "Characterizing serverless platforms with serverlessbench," in *Proceedings of SoCC*, 2020.

[57] N. Mahmoudi and H. Khazaei, "Simfaas: A performance simulator for serverless computing platforms," *CoRR*, vol. abs/2102.08904, 2021.

[58] E. Jonas, S. Venkataraman, I. Stoica, and B. Recht, "Occupy the Cloud: Distributed Computing for the 99%," *CoRR*, vol. abs/1702.04024, 2017.

[59] J. Spillner, "Snafu: Function-as-a-Service (FaaS) Runtime Design and Implementation," *CoRR*, vol. abs/1703.07562, 2017.

[60] J. Kuhlenkamp, S. Werner, M. C. Borges, K. E. Tal, and S. Tai, "An evaluation of FaaS platforms as a foundation for serverless big data processing," in *Proceedings of UCC*, 2019.

[61] P. Vahidinia, B. Farahani, and F. S. Aliee, "Cold start in serverless computing: Current trends and mitigation strategies," in *Proc. of COINS*, 2020.

[62] I. Baldini, P. Cheng, S. J. Fink, N. Mitchell, V. Muthusamy, R. Rabbah, P. Suter, and O. Tardieu, "The serverless trilemma: function composition for serverless computing," in *Proceedings of ACM SIGPLAN*, 2017.

[63] T. Pfandzelter and D. Bermbach, "tinyfaas: A lightweight faas platform for edge environments," in *Proceedings of ICFC*, 2020.

[64] L. A. Barba, "Praxis of reproducible computational science," *Computing in Science & Engineering*, vol. 21, no. 1, pp. 73–78, 2019.

Acknowledgment

Many thanks to Tobias Heckel and Martin Endreß contributing to the research prototype. It was a lot of fun to do a research project in winter term 2018/2019 with you and also writing the cold start paper ([1]).

Also Martin's bachelor thesis investigated an interesting aspect, how we can compare different execution environments and use this information to predict to some extent the execution behavior on machine A while executing the function on machine B. Part of his work is also included in the calibration, simulation and mapping part of the research prototype. The interested reader is referred to our IEEE CLOUD publication ([9]).

A User Management

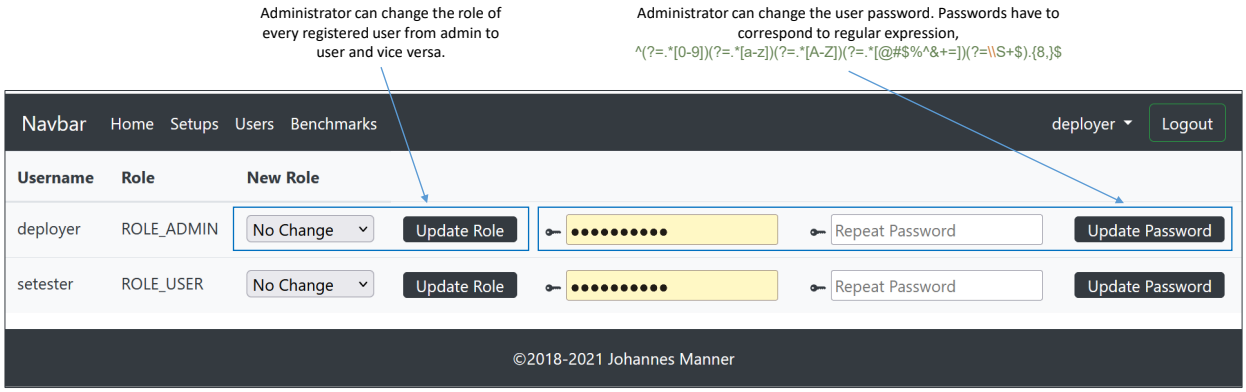


Figure 16: Users Page for Administrators to Change Roles and Passwords.

List of previous University of Bamberg reports

Bamberger Beiträge zur Wirtschaftsinformatik

Nr. 1 (1989) Augsburger W., Bartmann D., Sinz E.J.: Das Bamberger Modell: Der Diplom-Studiengang Wirtschaftsinformatik an der Universität Bamberg (Nachdruck Dez. 1990)

Nr. 2 (1990) Esswein W.: Definition, Implementierung und Einsatz einer kompatiblen Datenbankschnittstelle für PROLOG

Nr. 3 (1990) Augsburger W., Rieder H., Schwab J.: Endbenutzerorientierte Informationsgewinnung aus numerischen Daten am Beispiel von Unternehmenskennzahlen

Nr. 4 (1990) Ferstl O.K., Sinz E.J.: Objektmodellierung betrieblicher Informationsmodelle im Semantischen Objektmodell (SOM) (Nachdruck Nov. 1990)

Nr. 5 (1990) Ferstl O.K., Sinz E.J.: Ein Vorgehensmodell zur Objektmodellierung betrieblicher Informationssysteme im Semantischen Objektmodell (SOM)

Nr. 6 (1991) Augsburger W., Rieder H., Schwab J.: Systemtheoretische Repräsentation von Strukturen und Bewertungsfunktionen über zeitabhängigen betrieblichen numerischen Daten

Nr. 7 (1991) Augsburger W., Rieder H., Schwab J.: Wissensbasiertes, inhaltsorientiertes Retrieval statistischer Daten mit EISREVU / Ein Verarbeitungsmodell für eine modulare Bewertung von Kennzahlenwerten für den Endanwender

Nr. 8 (1991) Schwab J.: Ein computergestütztes Modellierungssystem zur Kennzahlenbewertung

Nr. 9 (1992) Gross H.-P.: Eine semantiktreue Transformation vom Entity-Relationship-Modell in das Strukturierte Entity-Relationship-Modell

Nr. 10 (1992) Sinz E.J.: Datenmodellierung im Strukturierten Entity-Relationship-Modell (SERM)

Nr. 11 (1992) Ferstl O.K., Sinz E. J.: Glossar zum Begriffsystem des Semantischen Objektmodells

Nr. 12 (1992) Sinz E. J., Popp K.M.: Zur Ableitung der Grobstruktur des konzeptuellen Schemas aus dem Modell der betrieblichen Diskurswelt

Nr. 13 (1992) Esswein W., Locarek H.: Objektorientierte Programmierung mit dem Objekt-Rollenmodell

Nr. 14 (1992) Esswein W.: Das Rollenmodell der Organsiation: Die Berücksichtigung aufbauorganisatorische Regelungen in Unternehmensmodellen

Nr. 15 (1992) Schwab H. J.: EISREVU-Modellierungssystem. Benutzerhandbuch

Nr. 16 (1992) Schwab K.: Die Implementierung eines relationalen DBMS nach dem Client/Server-Prinzip

Nr. 17 (1993) Schwab K.: Konzeption, Entwicklung und Implementierung eines computergestützten Bürovorgangssystems zur Modellierung von Vorgangsklassen und Abwicklung und Überwachung von Vorgängen. Dissertation

Nr. 18 (1993) Ferstl O.K., Sinz E.J.: Der Modellierungsansatz des Semantischen Objektmodells

Nr. 19 (1994) Ferstl O.K., Sinz E.J., Amberg M., Hagemann U., Malischewski C.: Tool-Based Business Process Modeling Using the SOM Approach

Nr. 20 (1994) Ferstl O.K., Sinz E.J.: From Business Process Modeling to the Specification of Distributed Business Application Systems - An Object-Oriented Approach -. 1st edition, June 1994

Ferstl O.K., Sinz E.J. : Multi-Layered Development of Business Process Models and Distributed Business Application Systems - An Object-Oriented Approach -. 2nd edition, November 1994

Nr. 21 (1994) Ferstl O.K., Sinz E.J.: Der Ansatz des Semantischen Objektmodells zur Modellierung von Geschäftsprozessen

Nr. 22 (1994) Augsburger W., Schwab K.: Using Formalism and Semi-Formal Constructs for Modeling Information Systems

Nr. 23 (1994) Ferstl O.K., Hagemann U.: Simulation hierarischer objekt- und transaktionsorientierter Modelle

Nr. 24 (1994) Sinz E.J.: Das Informationssystem der Universität als Instrument zur zielgerichteten Lenkung von Universitätsprozessen

Nr. 25 (1994) Wittke M., Mekinic, G.: Kooperierende Informationsräume. Ein Ansatz für verteilte Führungsinformationssysteme

Nr. 26 (1995) Ferstl O.K., Sinz E.J.: Re-Engineering von Geschäftsprozessen auf der Grundlage des SOM-Ansatzes

Nr. 27 (1995) Ferstl, O.K., Mannmeusel, Th.: Dezentrale Produktionslenkung. Erscheint in CIM-Management 3/1995

Nr. 28 (1995) Ludwig, H., Schwab, K.: Integrating cooperation systems: an event-based approach

Nr. 30 (1995) Augsburger W., Ludwig H., Schwab K.: Koordinationsmethoden und -werkzeuge bei der computergestützten kooperativen Arbeit

Nr. 31 (1995) Ferstl O.K., Mannmeusel T.: Gestaltung industrieller Geschäftsprozesse

Nr. 32 (1995) Gunzenhäuser R., Duske A., Ferstl O.K., Ludwig H., Mekinic G., Rieder H., Schwab H.-J., Schwab K., Sinz E.J., Wittke M: Festschrift zum 60. Geburtstag von Walter Augsburger

Nr. 33 (1995) Sinz, E.J.: Kann das Geschäftsprozeßmodell der Unternehmung das unternehmensweite Datenschema ablösen?

Nr. 34 (1995) Sinz E.J.: Ansätze zur fachlichen Modellierung betrieblicher Informationssysteme - Entwicklung, aktueller Stand und Trends -

Nr. 35 (1995) Sinz E.J.: Serviceorientierung der Hochschulverwaltung und ihre Unterstützung durch workflow-orientierte Anwendungssysteme

Nr. 36 (1996) Ferstl O.K., Sinz, E.J., Amberg M.: Stichwörter zum Fachgebiet Wirtschaftsinformatik. Erscheint in: Broy M., Spaniol O. (Hrsg.): Lexikon Informatik und Kommunikationstechnik, 2. Auflage, VDI-Verlag, Düsseldorf 1996

Nr. 37 (1996) Ferstl O.K., Sinz E.J.: Flexible Organizations Through Object-oriented and Transaction-oriented Information Systems, July 1996

Nr. 38 (1996) Ferstl O.K., Schäfer R.: Eine Lernumgebung für die betriebliche Aus- und Weiterbildung on demand, Juli 1996

Nr. 39 (1996) Hazebrouck J.-P.: Einsatzpotentiale von Fuzzy-Logic im Strategischen Management dargestellt an Fuzzy-System-Konzepten für Portfolio-Ansätze

Nr. 40 (1997) Sinz E.J.: Architektur betrieblicher Informationssysteme. In: Rechenberg P., Pomberger G. (Hrsg.): Handbuch der Informatik, Hanser-Verlag, München 1997

Nr. 41 (1997) Sinz E.J.: Analyse und Gestaltung universitärer Geschäftsprozesse und Anwendungssysteme. Angenommen für: Informatik '97. Informatik als Innovationsmotor. 27. Jahrestagung der Gesellschaft für Informatik, Aachen 24.-26.9.1997

Nr. 42 (1997) Ferstl O.K., Sinz E.J., Hammel C., Schlitt M., Wolf S.: Application Objects – fachliche Bausteine für die Entwicklung komponentenbasierter Anwendungssysteme. Angenommen für: HMD – Theorie und Praxis der Wirtschaftsinformatik. Schwerpunkheft ComponentWare, 1997

Nr. 43 (1997): Ferstl O.K., Sinz E.J.: Modeling of Business Systems Using the Semantic Object Model (SOM) – A Methodological Framework - . Accepted for: P. Bernus, K. Mertins, and G. Schmidt (ed.): Handbook on Architectures of Information Systems. International Handbook on Information Systems, edited by Bernus P., Blazewicz J., Schmidt G., and Shaw M., Volume I, Springer 1997

Ferstl O.K., Sinz E.J.: Modeling of Business Systems Using (SOM), 2nd Edition. Appears in: P. Bernus, K. Mertins, and G. Schmidt (ed.): Handbook on Architectures of Information Systems. International Handbook on Information Systems, edited by Bernus P., Blazewicz J., Schmidt G., and Shaw M., Volume I, Springer 1998

Nr. 44 (1997) Ferstl O.K., Schmitz K.: Zur Nutzung von Hypertextkonzepten in Lernumgebungen. In: Conradi H., Kreutz R., Spitzer K. (Hrsg.): CBT in der Medizin – Methoden, Techniken, Anwendungen -. Proceedings zum Workshop in Aachen 6. – 7. Juni 1997. 1. Auflage Aachen: Verlag der Augustinus Buchhandlung

Nr. 45 (1998) Ferstl O.K.: Datenkommunikation. In. Schulte Ch. (Hrsg.): Lexikon der Logistik, Oldenbourg-Verlag, München 1998

Nr. 46 (1998) Sinz E.J.: Prozeßgestaltung und Prozeßunterstützung im Prüfungswesen. Erschienen in: Proceedings Workshop „Informationssysteme für das Hochschulmanagement". Aachen, September 1997

Nr. 47 (1998) Sinz, E.J.:, Wismans B.: Das „Elektronische Prüfungsamt". Erscheint in: Wirtschaftswissenschaftliches Studium WiSt, 1998

Nr. 48 (1998) Haase, O., Henrich, A.: A Hybrid Respresentation of Vague Collections for Distributed Object Management Systems. Erscheint in: IEEE Transactions on Knowledge and Data Engineering

Nr. 49 (1998) Henrich, A.: Applying Document Retrieval Techniques in Software Engineering Environments. In: Proc. International Conference on Database and Expert Systems

Applications. (DEXA 98), Vienna, Austria, Aug. 98, pp. 240-249, Springer, Lecture Notes in Computer Sciences, No. 1460

Nr. 50 (1999) Henrich, A., Jamin, S.: On the Optimization of Queries containing Regular Path Expressions. Erscheint in: Proceedings of the Fourth Workshop on Next Generation Information Technologies and Systems (NGITS'99), Zikhron-Yaakov, Israel, July, 1999 (Springer, Lecture Notes)

Nr. 51 (1999) Haase O., Henrich, A.: A Closed Approach to Vague Collections in Partly Inaccessible Distributed Databases. Erscheint in: Proceedings of the Third East-European Conference on Advances in Databases and Information Systems – ADBIS'99, Maribor, Slovenia, September 1999 (Springer, Lecture Notes in Computer Science)

Nr. 52 (1999) Sinz E.J., Böhnlein M., Ulbrich-vom Ende A.: Konzeption eines Data Warehouse-Systems für Hochschulen. Angenommen für: Workshop „Unternehmen Hochschule" im Rahmen der 29. Jahrestagung der Gesellschaft für Informatik, Paderborn, 6. Oktober 1999

Nr. 53 (1999) Sinz E.J.: Konstruktion von Informationssystemen. Der Beitrag wurde in geringfügig modifizierter Fassung angenommen für: Rechenberg P., Pomberger G. (Hrsg.): Informatik-Handbuch. 2., aktualisierte und erweiterte Auflage, Hanser, München 1999

Nr. 54 (1999) Herda N., Janson A., Reif M., Schindler T., Augsburger W.: Entwicklung des Intranets SPICE: Erfahrungsbericht einer Praxiskooperation.

Nr. 55 (2000) Böhnlein M., Ulbrich-vom Ende A.: Grundlagen des Data Warehousing. Modellierung und Architektur

Nr. 56 (2000) Freitag B, Sinz E.J., Wismans B.: Die informationstechnische Infrastruktur der Virtuellen Hochschule Bayern (vhb). Angenommen für Workshop "Unternehmen Hochschule 2000" im Rahmen der Jahrestagung der Gesellschaft f. Informatik, Berlin 19. - 22. September 2000

Nr. 57 (2000) Böhnlein M., Ulbrich-vom Ende A.: Developing Data Warehouse Structures from Business Process Models.

Nr. 58 (2000) Knobloch B.: Der Data-Mining-Ansatz zur Analyse betriebswirtschaftlicher Daten.

Nr. 59 (2001) Sinz E.J., Böhnlein M., Plaha M., Ulbrich-vom Ende A.: Architekturkonzept eines verteilten Data-Warehouse-Systems für das Hochschulwesen. Angenommen für: WI-IF 2001, Augsburg, 19.-21. September 2001

Nr. 60 (2001) Sinz E.J., Wismans B.: Anforderungen an die IV-Infrastruktur von Hochschulen. Angenommen für: Workshop „Unternehmen Hochschule 2001" im Rahmen der Jahrestagung der Gesellschaft für Informatik, Wien 25. – 28. September 2001

Änderung des Titels der Schriftenreihe *Bamberger Beiträge zur Wirtschaftsinformatik* in *Bamberger Beiträge zur Wirtschaftsinformatik und Angewandten Informatik* ab Nr. 61

Note: The title of our technical report series has been changed from *Bamberger Beiträge zur Wirtschaftsinformatik* to *Bamberger Beiträge zur Wirtschaftsinformatik und Angewandten Informatik* starting with TR No. 61

Bamberger Beiträge zur Wirtschaftsinformatik und Angewandten Informatik

Nr. 61 (2002) Goré R., Mendler M., de Paiva V. (Hrsg.): Proceedings of the International Workshop on Intuitionistic Modal Logic and Applications (IMLA 2002), Copenhagen, July 2002.

Nr. 62 (2002) Sinz E.J., Plaha M., Ulbrich-vom Ende A.: Datenschutz und Datensicherheit in einem landesweiten Data-Warehouse-System für das Hochschulwesen. Erscheint in: Beiträge zur Hochschulforschung, Heft 4-2002, Bayerisches Staatsinstitut für Hochschulforschung und Hochschulplanung, München 2002

Nr. 63 (2005) Aguado, J., Mendler, M.: Constructive Semantics for Instantaneous Reactions

Nr. 64 (2005) Ferstl, O.K.: Lebenslanges Lernen und virtuelle Lehre: globale und lokale Verbesserungspotenziale. Erschienen in: Kerres, Michael; Keil-Slawik, Reinhard (Hrsg.); Hochschulen im digitalen Zeitalter: Innovationspotenziale und Strukturwandel, S. 247 – 263; Reihe education quality forum, herausgegeben durch das Centrum für eCompetence in Hochschulen NRW, Band 2, Münster/New York/München/Berlin: Waxmann 2005

Nr. 65 (2006) Schönberger, Andreas: Modelling and Validating Business Collaborations: A Case Study on RosettaNet

Nr. 66 (2006) Markus Dorsch, Martin Grote, Knut Hildebrandt, Maximilian Röglinger, Matthias Sehr, Christian Wilms, Karsten Loesing, and Guido Wirtz: Concealing Presence Information in Instant Messaging Systems, April 2006

Nr. 67 (2006) Marco Fischer, Andreas Grünert, Sebastian Hudert, Stefan König, Kira Lenskaya, Gregor Scheithauer, Sven Kaffille, and Guido Wirtz: Decentralized Reputation Management for Cooperating Software Agents in Open Multi-Agent Systems, April 2006

Nr. 68 (2006) Michael Mendler, Thomas R. Shiple, Gérard Berry: Constructive Circuits and the Exactness of Ternary Simulation

Nr. 69 (2007) Sebastian Hudert: A Proposal for a Web Services Agreement Negotiation Protocol Framework . February 2007

Nr. 70 (2007) Thomas Meins: Integration eines allgemeinen Service-Centers für PC-und Medientechnik an der Universität Bamberg – Analyse und Realisierungs-Szenarien. February 2007 (out of print)

Nr. 71 (2007) Andreas Grünert: Life-cycle assistance capabilities of cooperating Software Agents for Virtual Enterprises. März 2007

Nr. 72 (2007) Michael Mendler, Gerald Lüttgen: Is Observational Congruence on μ-Expressions Axiomatisable in Equational Horn Logic?

Nr. 73 (2007) Martin Schissler: out of print

Nr. 74 (2007) Sven Kaffille, Karsten Loesing: Open chord version 1.0.4 User's Manual. Bamberger Beiträge zur Wirtschaftsinformatik und Angewandten Informatik Nr. 74, Bamberg University, October 2007. ISSN 0937-3349.

Nr. 75 (2008) Karsten Loesing (Hrsg.): Extended Abstracts of the Second *Privacy Enhancing Technologies Convention* (PET-CON 2008.1). Bamberger Beiträge zur Wirtschaftsinformatik und Angewandten Informatik Nr. 75, Bamberg University, April 2008. ISSN 0937-3349.

Nr. 76 (2008) Gregor Scheithauer, Guido Wirtz: Applying Business Process Management Systems – A Case Study. Bamberger Beiträge zur Wirtschaftsinformatik und Angewandten Informatik Nr. 76, Bamberg University, May 2008. ISSN 0937-3349.

Nr. 77 (2008) Michael Mendler, Stephan Scheele: Towards Constructive Description Logics for Abstraction and Refinement. Bamberger Beiträge zur Wirtschaftsinformatik und Angewandten Informatik Nr. 77, Bamberg University, September 2008. ISSN 0937-3349.

Nr. 78 (2008) Gregor Scheithauer, Matthias Winkler: A Service Description Framework for Service Ecosystems. Bamberger Beiträge zur Wirtschaftsinformatik und Angewandten Informatik Nr. 78, Bamberg University, October 2008. ISSN 0937-3349.

Nr. 79 (2008) Christian Wilms: Improving the Tor Hidden Service Protocol Aiming at Better Performances. Bamberger Beiträge zur Wirtschaftsinformatik und Angewandten Informatik Nr. 79, Bamberg University, November 2008. ISSN 0937-3349.

Nr. 80 (2009) Thomas Benker, Stefan Fritzemeier, Matthias Geiger, Simon Harrer, Tristan Kessner, Johannes Schwalb, Andreas Schönberger, Guido Wirtz: QoS Enabled B2B Integration. Bamberger Beiträge zur Wirtschaftsinformatik und Angewandten Informatik Nr. 80, Bamberg University, May 2009. ISSN 0937-3349.

Nr. 81 (2009) Ute Schmid, Emanuel Kitzelmann, Rinus Plasmeijer (Eds.): Proceedings of the ACM SIGPLAN Workshop on *Approaches and Applications of Inductive Programming* (AAIP'09), affiliated with ICFP 2009, Edinburgh, Scotland, September 2009. Bamberger Beiträge zur Wirtschaftsinformatik und Angewandten Informatik Nr. 81, Bamberg University, September 2009. ISSN 0937-3349.

Nr. 82 (2009) Ute Schmid, Marco Ragni, Markus Knauff (Eds.): Proceedings of the KI 2009 Workshop *Complex Cognition*, Paderborn, Germany, September 15, 2009. Bamberger Beiträge zur Wirtschaftsinformatik und Angewandten Informatik Nr. 82, Bamberg University, October 2009. ISSN 0937-3349.

Nr. 83 (2009) Andreas Schönberger, Christian Wilms and Guido Wirtz: A Requirements Analysis of Business-to-Business Integration. Bamberger Beiträge zur Wirtschaftsinformatik und Angewandten Informatik Nr. 83, Bamberg University, December 2009. ISSN 0937-3349.

Nr. 84 (2010) Werner Zirkel, Guido Wirtz: A Process for Identifying Predictive Correlation Patterns in Service Management Systems. Bamberger Beiträge zur Wirtschaftsinformatik und Angewandten Informatik Nr. 84, Bamberg University, February 2010. ISSN 0937-3349.

Nr. 85 (2010) Jan Tobias Mühlberg und Gerald Lüttgen: Symbolic Object Code Analysis. Bamberger Beiträge zur Wirtschaftsinformatik und Angewandten Informatik Nr. 85, Bamberg University, February 2010. ISSN 0937-3349.

Nr. 86 (2010) Werner Zirkel, Guido Wirtz: Proaktives Problem Management durch Eventkorrelation – ein *Best Practice* Ansatz. Bamberger Beiträge zur Wirtschaftsinformatik und Angewandten Informatik Nr. 86, Bamberg University, August 2010. ISSN 0937-3349.

Nr. 87 (2010) Johannes Schwalb, Andreas Schönberger: Analyzing the Interoperability of WS-Security and WS-ReliableMessaging Implementations. Bamberger Beiträge zur Wirtschaftsinformatik und Angewandten Informatik Nr. 87, Bamberg University, September 2010. ISSN 0937-3349.

Nr. 88 (2011) Jörg Lenhard: A Pattern-based Analysis of WS-BPEL and Windows Workflow. Bamberger Beiträge zur Wirtschaftsinformatik und Angewandten Informatik Nr. 88, Bamberg University, March 2011. ISSN 0937-3349.

Nr. 89 (2011) Andreas Henrich, Christoph Schlieder, Ute Schmid [eds.]: Visibility in Information Spaces and in Geographic Environments – Post-Proceedings of the KI'11 Workshop. Bamberger Beiträge zur Wirtschaftsinformatik und Angewandten Informatik Nr. 89, Bamberg University, December 2011. ISSN 0937-3349.

Nr. 90 (2012) Simon Harrer, Jörg Lenhard: Betsy - A BPEL Engine Test System. Bamberger Beiträge zur Wirtschaftsinformatik und Angewandten Informatik Nr. 90, Bamberg University, July 2012. ISSN 0937-3349.

Nr. 91 (2013) Michael Mendler, Stephan Scheele: On the Computational Interpretation of CKn for Contextual Information Processing - Ancillary Material. Bamberger Beiträge zur Wirtschaftsinformatik und Angewandten Informatik Nr. 91, Bamberg University, May 2013. ISSN 0937-3349.

Nr. 92 (2013) Matthias Geiger: BPMN 2.0 Process Model Serialization Constraints. Bamberger Beiträge zur Wirtschaftsinformatik und Angewandten Informatik Nr. 92, Bamberg University, May 2013. ISSN 0937-3349.

Nr. 93 (2014) Cedric Röck, Simon Harrer: Literature Survey of Performance Benchmarking Approaches of BPEL Engines. Bamberger Beiträge zur Wirtschaftsinformatik und Angewandten Informatik Nr. 93, Bamberg University, May 2014. ISSN 0937-3349.

Nr. 94 (2014) Joaquin Aguado, Michael Mendler, Reinhard von Hanxleden, Insa Fuhrmann: Grounding Synchronous Deterministic Concurrency in Sequential Programming. Bamberger Beiträge zur Wirtschaftsinformatik und Angewandten Informatik Nr. 94, Bamberg University, August 2014. ISSN 0937-3349.

Nr. 95 (2014) Michael Mendler, Bruno Bodin, Partha S Roop, Jia Jie Wang: WCRT for Synchronous Programs: Studying the Tick Alignment Problem. Bamberger Beiträge zur Wirtschaftsinformatik und Angewandten Informatik Nr. 95, Bamberg University, August 2014. ISSN 0937-3349.

Nr. 96 (2015) Joaquin Aguado, Michael Mendler, Reinhard von Hanxleden, Insa Fuhrmann: Denotational Fixed-Point Semantics for Constructive Scheduling of Synchronous Concurrency. Bamberger Beiträge zur Wirtschaftsinformatik und Angewandten Informatik Nr. 96, Bamberg University, April 2015. ISSN 0937-3349.

Nr. 97 (2015) Thomas Benker: Konzeption einer Komponentenarchitektur für prozessorientierte OLTP- & OLAP-Anwendungssysteme. Bamberger Beiträge zur Wirtschaftsinformatik und Angewandten Informatik Nr. 97, Bamberg University, Oktober 2015. ISSN 0937-3349.

Nr. 98 (2016) Sascha Fendrich, Gerald Lüttgen: A Generalised Theory of Interface Automata, Component Compatibility and Error. Bamberger Beiträge zur Wirtschaftsinformatik und Angewandten Informatik Nr. 98, Bamberg University, March 2016. ISSN 0937-3349.

Nr. 99 (2014) Christian Preißinger, Simon Harrer: Static Analysis Rules of the BPEL Specification: Tagging, Formalization and Tests. Bamberger Beiträge zur Wirtschaftsinformatik und Angewandten Informatik Nr. 99, Bamberg University, August 2014. ISSN 0937-3349.

Nr. 100 (2016) Cedrik Röck, Stefan Kolb: Nucleus - Unified Deployment and Management for Platform as a Service. Bamberger Beiträge zur Wirtschaftsinformatik und Angewandten Informatik Nr. 100, Bamberg University, March 2016. ISSN 0937-3349.

Nr. 101 (2016) Michael Mendler, Partha S. Roop, Bruno Bodin: A Novel WCET Semantics of Synchronous Programs. Bamberger Beiträge zur Wirtschaftsinformatik und Angewandten Informatik Nr. 101, Bamberg University, June 2016. ISSN 0937-3349.

Nr. 102 (2017) Joaquín Aguado, Michael Mendler, Marc Pouzet, Partha Roop, Reinhard von Hanxleden: Clock-Synchronised Shared Objects for Deterministic Concurrency. Bamberger Beiträge zur Wirtschaftsinformatik und Angewandten Informatik Nr. 102, Bamberg University, July 2017. ISSN 0937-3349.

Nr. 103 (2018) Eugene Yip, Erjola Lalo, Gerald Lüttgen, Michael Deubzer, Andreas Sailer: Optimized Buffering of Time-Triggered Automotive Software. Bamberger Beiträge zur Wirtschaftsinformatik und Angewandten Informatik Nr. 103, Bamberg University, September 2018. ISSN 0937-3349.

Nr. 104 (2018) Daniel Hallmann: Die COCOMO-Modelle im Licht der agilen Softwareentwicklung. Bamberger Beiträge zur Wirtschaftsinformatik und Angewandten Informatik Nr. 104, Bamberg University, October 2018. ISSN 0937-3349.

Nr. 105 (2021) Johannes Manner: SeMoDe – Simulation and Benchmarking Pipeline for Function as a Service. Bamberger Beiträge zur Wirtschaftsinformatik und Angewandten Informatik Nr. 105, Bamberg University, November 2021. ISSN 0937-3349.