UNIVERSITY OF MISKOLC

FACULTY OF MECHANICAL ENGINEERING AND INFORMATICS

DISSECT-CF-FaaS: A Simulation Environment for Simulating Functions-as-a-Service

SUMMARY OF PHD THESIS

Written by:
Dilshad Hassan Sallo

MSc in Advanced Computer Science with a Specialisation in Software Technology

'JÓZSEF HATVANY' DOCTORAL SCHOOL
OF INFORMATION SCIENCE, ENGINEERING AND TECHNOLOGY

HEAD Of DOCTORAL SCHOOL
Prof. Dr. Jenő SZIGETI

ACADEMIC SUPERVISOR
Prof. Dr. Gábor Kecskeméti

Miskolc, 2024

Contents

1 Introduction

Serverless is a new computing paradigm adopted by several cloud providers. It provides a new style of delivering cloud services by letting a user to mainly concentrate on coding rather than managing backend-infrastructure and operations. In most cases, commercial providers are not always the most favourable choices for researchers to execute and evaluate their desired scenarios, due to having a costly and complex environment (leading to non-reproducible results).

One alternative solution is simulators, which were opted by the research communities to evaluate scenarios in reduced-cost and easy-to-setup-environments. Over the last years, numerous cloud simulators have been built to support the IaaS model. They offer a flexible environment to experiment on various algorithms and scenarios in the field of infrastructure management. To obtain respectable precision, simulators use real traces often collected and offered by commercial providers. These traces represent comprehensive information about executed tasks reflecting users' behaviour within providers.

Despite the widespread use of cloud simulators, they are still mainly focused on supporting more traditional IaaS scenarios, and this reduces their applicability in the serverless and Functions-as-a-Service domains. There are several features essential to support the serverless models that are missing from most IaaS frameworks. For example, the need of simulating and execute multiple events in parallel. Moreover, workload traces typically employed by IaaS simulators are not well adoptable to the new computing model. Thus, they do not represent the new kind of users' behaviour. In addition to that, IaaS simulators are not designed to take responsibility for managing the necessary infrastructure, complex provisioning, and configurations on behalf of a user, which is how he/she deals with serverless systems.

The need for an integrated serverless environment capable of mimicking the behaviour of real providers is essential towards evaluating applications and scenarios reliant on the concepts of this computing paradigm. To fulfil the researchers' purpose, the environment has to support the newly introduced features, computing style, and resource constraints that led to exist of this computing type.

As serverless technology still based on underlying infrastructure that is abstracted from a user, it is beneficial to extend existing IaaS simulators to support serverless functionalities and features. The IaaS simulator called DISSECT-CF [Kec15] is selected to fulfill this purpose as it is extensible, allows sharing low-level computing, supports loading and managing several trace file formats, and its performance is significantly higher than most simulators in the field. All these reasons are leading us to the main aim of the research.

1.1 Research goals

This research aims to develop a comprehensive serverless environment on DISSECT-CF simulator. This new environment is capable of simulating and evaluating serverless applications and scenarios. The research is divided into three goals.

1. The ability to generate realistic serverless workloads close to real users' behaviour. Also supporting scaling such workloads to fit any researchers' desired scenario.

2. The ability to mimic real serverless providers in terms of provisioning resources policy, internal mechanism, estimating costs and provided services.

3. The ability to perform parallel execution for revealing the internal behaviour of large-scale simulation session.

2 Related work

Unfortunately, there are no established simulation frameworks that can supportresearch on the challenges accompanying serverless computing. The few serverlesssimulators that exist focus on specific functionality or aspects, but they could notcomprehensively support the above listed features,

Recently, few serverless simulators have been developed either by extending the IaaS simulators' functionalities or from scratch. To fulfil the researchers' purpose towards simulating FaaS, serverless simulators have to mimic a real provider by offering the foundations and features of this new technology. Unfortunately, serverless simulation is in its infancy, frameworks only partially support the features (e.g., auto-scaling) of the new model. Thus, using them for evaluating new approaches to manage FaaS systems could still lead to potentially misleading research results.

DFaaSCloud [JCSY19] simulator extends CloudSim [CRB+11] to support some serverless features. It allows defining FaaS functions with various profiles and characteristics that determine their behaviour during simulation. However, it has the following major limitations for its support of serverless environments: (*i*) establishing and managing the virtual infrastructure backing functions is not fully supported because it does not consider the providers' policies, (*ii*) unable to utilise large scale generated realistic serverless traces due to use synthetic workload, and (*iii*) extracting serverless performance metrics (e.g., probability of cold-start and average utilization) from the simulation are missing.

OpenDC serverless [Jou20] is a framework that was introduced based on the OpenDC simulator. It allows to model and test custom FaaS patterns. It introduced the essential architectural component, instance routing policy such as that imitates the basic serverless platform. However, it lacks an important concept of serverless,namely auto-scaling resources that responds to workload. It also doesn't provide information about the internal behaviour of infrastructure and function status.

SimFaaS [MK21] is a simulation platform that introduces a serverless environment to enable researchers to develop and optimise FaaS applications. It is designed to extract performance metrics from simulation. However, one of the missing features that make simFaaS, not-a-comprehensive-simulator, is the trigger, which defines how FaaS functions will invoke during simulation. SimFaaS also doesn't provide a foundation for a cost model of real providers. Moreover, the provider's resource constraints cannot be applied to simFaaS. Extracting performance metrics in simFaaS mainly relies on the already existing attributes of functions in the author's proprietary trace format, such as cold-start probability. Additionally, it calculates some metrics based on the traces, not actually evaluating the results from the simulation session.

Based on the above points, there is a lack of features in existing serverless simulators that are not covering this new computing paradigm's needs. We can conclude that the need to enrich the research community with a comprehensive serverless environment is crucial to fulfil researchers' expectations towards this technology, by offering the services behind serverless computing. Therefore, in this dissertation, we addressed the aforementioned limitations by introducing an integrated serverless environment (dubbed as DISSECT-CF-FaaS) able to generate and simulate large-scale serverless workloads. This environment is capable of supporting several real providers and mimicking their distinctive policies. It also provides parallel execution to foster analyzing the internal behaviour of our environment. The introduced serverless environment was built based on DISSECT-CF simulator. The rest of this dissertation is going to focus on these extensions.

3 Scientific Results

3.1 Generating Realistic Serverless Traces

Supporting serverless computing model by IaaS simulators necessitate the introduction of additional features. One of the significant features is to enable the simulator to generate realistic-trace and predict the behaviour of such realistic serverless workloads. We introduced our serverless architecture based on DISSECT-CF that is aimed at offering automated management of function-as-a-service workloads. The

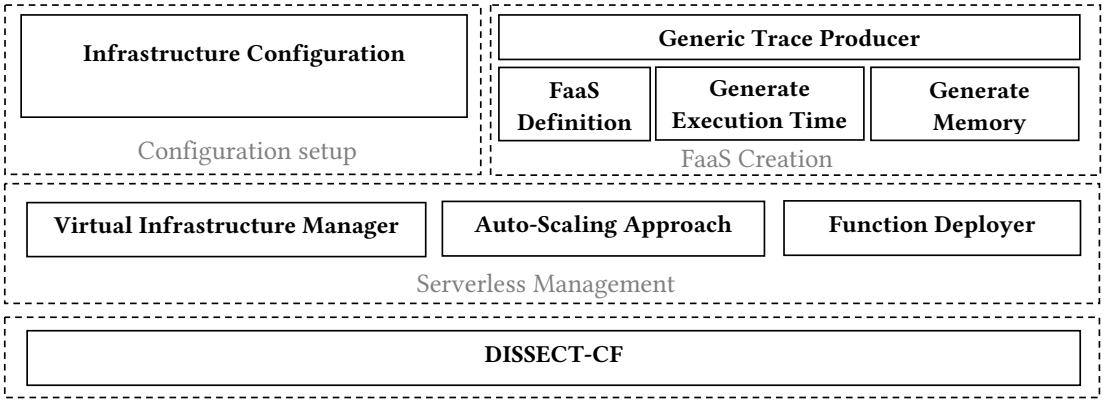


Figure 1: Architecture of proposed model

proposed architecture consists of three layers built on the top of DISSECT-CF, as shown in Figure 1.

Configuration setup layer handles the user options by establishing proper infrastructure based on the selected provider's plan. **FaaS creation layer** is responsible for generating serverless traces form a selected dataset file. **Serverless management layer** is responsible for managing virtual infrastructure (that backs the serverless computing platform), as well as providing just-enough resources for all the function invocations. Our approach are based on the Azure Functions dataset [SFG+20] and it generates execution time and memory utilisation values via percentile values that were provided on daily basis.

3.1.1 Evaluation of generator approach

To validate our approach that will act as a foundation for evaluation serverless model, we picked randomly 5000 functions and we have generated 5000 invocations for each function. For each function that has generated 5000 invocations, we calculated the percentile values and average for both execution time and memory utilization. Then, we measured the coefficient of determination (R^2) between the generated and original values to show data accuracy. For execution time, R^2 was 0.8438 for percentiles and 0.9956 for average. For memory, R^2 were 0.8999 and 0.9977 for percentiles and average, respectively.

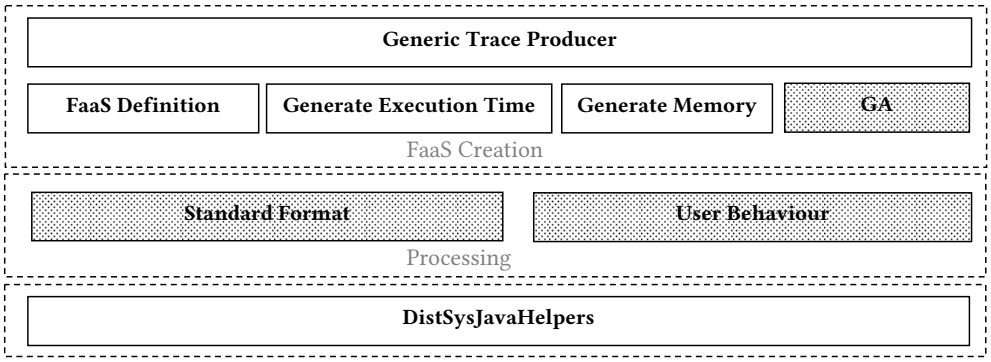


Figure 2: Architecture of generator approach

3.1.2 Improving our previous generator approach

In our previous work, we have shown that this simple approach was capable of providing realistic execution time and memory utilisation averages compared to the original dataset. However, the limitations of the previous approach were as follows: (*i*) The generated trace's percentiles for execution time and memory utilisation were not sufficiently close according to the coefficient of determination (R^2). (*ii*) The approach was limited to single invocations, realistic representation of multiple users was not supported. (*iii*) It was problematic to reuse the generated traces due to its strong link to DISSECT-CF.

3.1.3 Architecture of enhanced approach

To remedy the aforementioned limitations, we introduced independent architecture that holds FaaS creation layer from the previous approach as well as adding three further components, namely `GA`, `User Behaviour` and `Standard Format` as shown in Figure 2. They are introduced to improve quality, enabling scaling workload and providing reusability of generated traces respectively.

The previous approach generates one set of values (representing one individual in genetic algorithm concept) for each unique function and its invocations. Although our approach generates values within the range of percentiles, good values of (R^2) between the generated and original could not be produced by a single iteration as they are generated randomly. To address this issue, we introduced a genetic algorithm (`GA` component) to our previous approach. We produce several sets of values, and then it selects optimal values to constitute a single set for function and its invocations.

Scaling-workload is essential to support researchers' scenarios that require small

infrastructure to achieve their desired purposes, such as predicting the consumption behaviour of users to stimulate resource usage awareness. Therefore, we introduced `User Behavior` component that enables scaling workloads by detecting users' behaviour in terms of calculating the percentage of participation for each unique user in a dataset.

`Standard Format` component enables the reusability of generated traces by converting them to other formats as standalone traces, and getting rid of the process of generating traces repeatedly. When the serverless functions are generated, this component obtains each function's invocation with all its attributes (e.g., execution time, tasks' id and amount of memory) to store in its repository. Then, for each invocation, it arranges its attribute values based on the desired format, as each one has its own ordering. Finally, it goes through this process till the end of functions, then it produces standalone trace file.

3.1.4 Evaluation of improved percentiles

To validate our approach of generating realistic traces with the help of the genetic algorithm, we used uniform randomly generator to pick up 5000 functions, and we have generated 5000 invocations for each unique function based on the percentiles that disclosed in Azure dataset. We concluded with a very good result of R^2, which was 0.9994 for execution time and 0.9995 for memory utilisation.

3.1.5 Evaluation of users' behavior

To validate users' behaviour, we have chosen one day of the Azure dataset, which contains comprehensive functions that executed in the Azure provider. Then, we invoked `User Behavior` component to analyse the selected file statistically. The `User Behavior` component demonstrated that the file came with around 853 million invocations, 36,456 services and 8,590 users. Moreover, it provided detailed information regarding each user's invocation number and the percentage of participation.

We validated the scaling approach by producing different workload sizes that fit small and large infrastructure configurations. We, then, invoked `User Behavior` component for statistical analysis of each generated workload. Finally, we compared the percentage of users' participation in all different workloads, with the original dataset by using R^2 as shown in Table 1. We also measured R^2 between the average of percentiles for execution time and memory utilisation for all generated workloads against the original one to show data accuracy. The results show that our approach enables scaling workloads efficiently with the real users' behaviour. It also shows

Table 1: Scaling workloads with real users' behaviour

Workload Size	R^2 **(User's percentage)**	R^2 **(Execution time)**	R^2 **(Memory)**
10^3	0.9999	0.9969	0.9986
10^4	1	0.9986	0.9984
10^5	1	0.9993	0.9989
10^6	1	0.9993	0.9981
10^7	1	0.9995	0.9997
10^8	1	1	0.9998

that the generated execution time and memory utilisation percentiles resemble the original values during scaling workloads.

3.1.6 Evaluation of converting approach

Converting generated traces is beneficial to computing simulators that only support real traces and consider all the function's attributes in trace. To validate our converting approach, we have generated traces with different formats and simulated them by following simulators that belong to different fields, namely, DISSECT-CF (cloud simulator), GridSim (Grid simulator) and simFaaS (serverless simulator).

3.1.7 DISSECT-CF

One of the approaches to verify the reliability of converting generated traces is monitoring the internal behaviour of the simulation while simulating the same tasks in from different formats. Therefore, we have selected DISSECT-CF simulator as it is enabling observing the internal behaviour of the infrastructure during simulation as well as offering precise results. To show the accuracy of converting traces to other formats, we generated 20 thousand functions and we then asked to convert the same generated trace to Grid Workload Format (GWF) and Standard Workload Format(SWF) traces. First, we generated a workload from the Azure dataset (CSV) that was directly simulated by DISSECT-CF. In the second round, we simulated the transformed GWF and SWF traces.

We have observed the internal behaviour of the simulated infrastructure in terms of simulated timespan, number of used virtual machines, average utilization of physical machines and total power consumption for all experiments. Table 2 shows that simulation timespan and the number of used VMs are identical between the original trace (CSV) and converted traces (GWF and SWF). However, there is a difference of 1.3% and 0.08% for the average utilisation of PMs and total power consumption,

Table 2: Simulating 20 k functions with different formats

Metrics	**CSV**	**GWF**	**SWF**
Average utilization of PMs(%)	0.8119	0.8014	0.8014
Total power consumption (kWh)	73.5076	73.4451	73.4451
Simulated timespan (ms)	87792001	87792001	87792001
Number of used VMs	301	301	301

Table 3: Comparing simulating functions using DISSECT-CF and GridSim in terms of simulated timespan

Simulator	**1k**	**5k**	**10k**	**50k**	**100k**	**500k**	**1m**
DISSECT-CF	33.6s	2.79m	5.58m	27.9m	55.9m	4.65h	9.31h
GridSim	34.3s	2.81m	5.59m	27.9m	55.9m	4.65h	9.31h
Difference	1.95%	0.4%	0.2%	0.04%	0.02%	∼0%	∼0%

respectively. This meets our expectations and shows that our approach of converting is properly and realistic.

3.1.8 GridSim

To validate our approach of converting traces and demonstrating their usability by other simulators, we explored the most popular open-source simulators in the computing field that could use these traces. We have considered two factors while exploring these simulators. First, the recent year these simulators were used by researchers for evaluating real computing scenarios. Second, the types of workloads that are supported by these simulators. Based on these,

We have selected the recent and popular simulator GridSim [BM02] for simulating and validating our converting approach. GridSim considers all attributes of a task, such as execution time, submit time, memory utilisation and others, as the same as DISSECT-CF does. We have simulated different workload sizes starting from small-scale trace to large-scale using DISSECT-CF and GridSim. As both were set to the same configuration and have simulated identical workloads, the timespan will reflect the overall execution time of the simulation session. We have calculated the percent difference of simulated timespan for each workload, as shown in Table 3. The result shows the average percent difference is 0.37% between both simulators, which shows how accurately the traces are generated to match the original one and converted to standard formats.

3.1.9 SimFaaS

As our generator approach able to produce traces in different standard formats that meet a user's and a simulator requirements, we converted Azure dataset to AWS Lambda traces to be used as input to simFaaS simulator. Our model also provides performance metrics likes simFaaS, but it extracts them from simulation session. To check the accuracy of our model output, We generated 15 AWS Lambda traces from this Azure dataset. For each trace, we have randomly selected thousand unique functions with their invocations. As a result, the generated traces contain around 500 thousand requests. After that, we simulated these traces with our model and simFaaS framework to extract the performance metrics from the simulation.

To validate the converting approach, we measured R^2 of performance metrics for both. The results of R^2 were 0.9999, 0.9960, 0.9177, and 0.9525 for arrival rate, cold-start probability, average utilisation and average idle instances, respectively. This indicates the accuracy of our approach for producing traces that used by the simFaaS serverless simulator.

Thesis 1 **Related Publications: [5, 6]**

I proposed a novel approach for generating realistic serverless traces to enrich cloud computing simulators with varying characteristic workload types. My approach applies a genetic algorithm to produce and select the best generated functions' attributes that resemble the behaviour in a real-life dataset. It also enables scaling-workload to fit desired scenarios while maintaining the users' behaviour disclosed in the real-life dataset. Finally, it supports the reusability of the generated traces in other computing simulators by adapting the traces to popular formats.

3.2 An Extension of DISSECT-CF to Simulate Function-as-a-Service

In the research community, simulators are the most common environments for evaluating algorithms and scenarios, as they provide easy-setup, low-cost and reproducible environments. The foundation of modern computing technologies such as fog computing, edge computing, and serverless computing, are built on the concept of cloud computing. Thus, extending IaaS simulators to support other computing models is an essential step towards offering a versatile solution for the research community.

In subsection 3.1, we introduced a rudimentary model for serverless systems based on DISSECT-CF. However, it had noticeable limitations such as (*i*) focused only on Azure Functions providers, which limited its use for other providers and theirs policies, (*ii*) the quintessential concept widely used by FaaS systems is not offered,

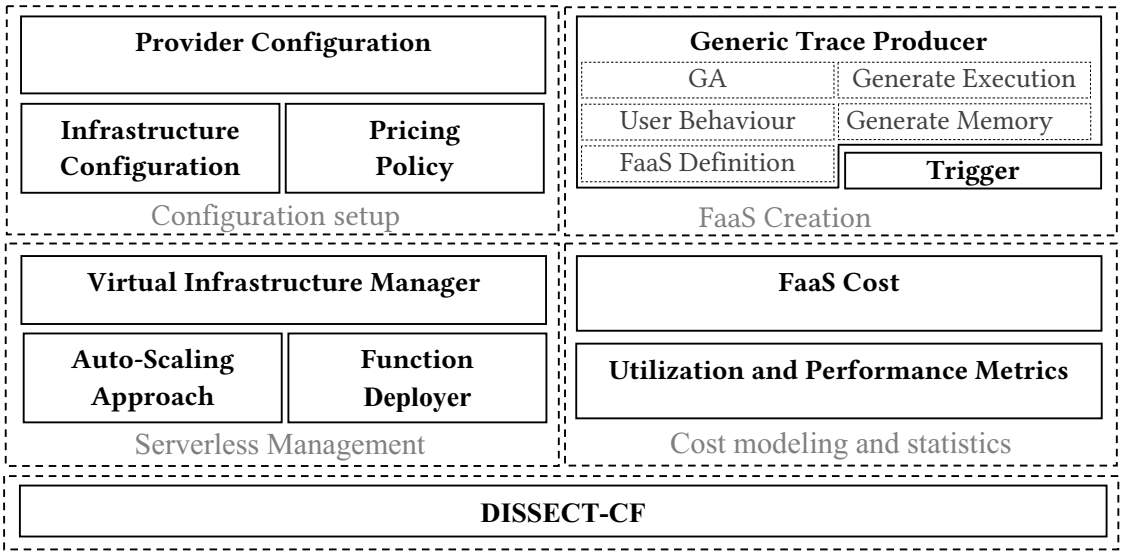


Figure 3: Architecture of our serverless environment

(*iii*) its cost model was not generic enough to support the newer providers, and (*iv*) essential, serverless focused performance metrics such as the number of provisioned instances at a specific time, are complicated to acquire.

3.2.1 Proposed architecture

To remedy the aforementioned limitations, we introduced new layers and components to the architecture of the previous model, in addition we updated the internal implementation of some components. This proposed architecture consists of four layers built on top of the core DISSECT-CF simulator (see Figure 3).

Configuration setup layer provides direct interaction with a researcher to establish underlying infrastructure and cost policies through provider selection and configuration. **FaaS Creation layer** focuses on the functionality running on top of the previously configured infrastructure. Primarily, this is done via the selection and customisation of a trace/dataset used for modelling serverless function behaviour. It also enables associating triggers to these functions. **Serverless management layer** represents the internal implementation that reflect the policy of serverless providers. This is achieved through the management of virtual infrastructures behind each serverless function created earlier. In out final layer, **Cost modeling and statistics**, we provide components to estimate the cost of the workload that passed through the simulation.

Table 4: Simulating 100k invocations using different memory sizes

Size	Cost($)	Average run time(ms)	Cold-start(%)	Warm-start(%)	Arrival Rate(s)
128	3.553	17329	1.059	98.941	0.0036
512	3.503	4321	0.614	99.386	0.0073
1024	3.517	2156	0.555	99.444	0.0103
2048	3.625	1114	0.523	99.477	0.0145
3072	3.504	717	0.510	99.490	0.0151

3.2.2 Evaluation

We evaluate the effectiveness of our extension by producing services that resemble the AWS Lambda and Azure Functions providers, as they are most popular serverless providers [RSBT19]. Moreover, mimicking the providers' policies in terms of resource constraints and associating triggers to serverless functions to reflect the realistic internal behaviour of serverless computing.

3.2.3 Evaluation of cost model

We have evaluated our generic cost-model by imitating AWS Lambda provider, and then we have conducted experiments to estimate the cost of 100k serverless function invocations but using different memory configurations. The Table 4 shows that we obtained similar cost for all experiments but with different running time as expected. The reason is AWS Lambda provider offers a user trade-off between cost and running time. Increasing memory size is costly but leads to reduced running time and cold-start probability. Thus, a user can have a fast or slow wall time with almost the same cost.

3.2.4 Evaluation of trigger

A trigger is used to provide different ways to invoke a function. To investigate the timer trigger in our serverless environment, we imitate the Azure provider's policy and we then generated varying sized realistic traces and conducted an experiment that involves five different groups of functions. Each group consists of 500 types of functions and has different invocation intervals determined by their triggers as shown in the Table 5. We ran the functions of all groups for one simulated day. The result shows that the probability of cold-start for the group of functions that has the smallest interval is less than for other groups. As our environment observes the status of instances and measures their life-time, it allows more frequent reuse of their instances than those belonging to other function groups.

Table 5: Using timer trigger for five groups of functions with different trigger intervals

	Interval (s)	Prob. of cold-start (%)	Prob. of warm-start (%)	No invocations
Group1	30	0.1945	99.8054	1440000
Group2	60	0.2461	99.7538	720000
Group3	300	1.0277	98.9722	144000
Group4	600	1.1013	98.8986	72000
Group5	1200	2.3555	97.6444	36000

Table 6: Average performance metrics of the instances were extracted while simulating different workload sizes

Workload size	Concurrent no	Lifetime(s)	Running-time(s)	Idle-time(s)
1k	65	1239	40	1199
10k	67	1381	197	1183
100k	98	1481	323	1158
200k	155	1541	408	1133

3.2.5 Evaluation of performance metrics

Our serverless environment enables extracting essential performance metrics to reveal the internal behaviour of the imitated provider in terms of applying its policy, constraints and backend provisioning resources for understanding internal mechanism. We have demonstrated our performance metrics through simulating different workloads sized and we then simulated this workload by mimicking the Azure Functions provider. The result shows that our serverless environment provisioned more instances when the workload size was heavy, as shown in Table 6. The reason is Azure Functions provider can scale up to 200 instances concurrently per function-app type. The results also show that the utilisation of instances (running time) is increased when workload size is heavy.

Thesis 2. **Related Publications:** [2, 3, 5, 7]

I proposed a comprehensive serverless extension (DISSECT-CF-FaaS) to the research community for evaluating a wide range of real-case FaaS scenarios in an environment that imitates commercial providers' behaviour. This environment is capable of capturing real behaviour services to enable establishing a cost model, offering scaling up-down function instances, introducing a trigger mechanism comparable to real usage behaviour, and applying constrained on provisioning resources. It also extracts performance metrics from the simulation session to reveal how the internal behaviour of provisioning resources responds while serverless functions are simulated.

3.3 Parallel Event System to Reveal the Internal Behavior of our Serverless Environment

Most cutting-edge technologies such as serverless computing have designed to allow multiple events such as serverless functions (FaaSs) and their invocations or computing tasks in cloud, happen at the same time to obtain better performance. Its paradigm encourages users to execute events in a parallel by managing the backend infrastructure on their behalf. When we come to simulation environment to mimic advanced technologies mechanism, the architecture of simulator is critical word to demonstrate its capability towards parallelism. Although, the selected simulator DISSECT-CF reduces the execution time of equal quality/detail simulations done compared to several other simulators in the field, it still does so in a sequential fashion. Therefore, we aim to set the foundations to support simulations that require high performance.

Based on the existing API of DISSECT-CF, parallelisation could happen for executing of simultaneously happening events (i.e., events that should happen in the same time instance or tick of a simulation) in the event system of DISSECT-CF.

3.3.1 The parallelisation of simultaneous events

We introduced the new inner `Parallel` class to the event system of DISSECT-CF to apply parallelism. Our approach includes two algorithms. Algorithm 1 determines the need of executing list of events in sequential or parallel fashion based on the number of events.

Algorithm 1 Determining the need for parallelism

```
1: threshold = specified size
2: list = all simultaneous events
3: if list.size ⩽ threshold then
4:     while list.notEmpty do
5:         event = get single event from list
6:         Execute event
7:     end while
8: else
9:     execute Parallel(list.lowIndex, list.upperIndex)
10: end if
```

After the decision to parallelise, the actual parallelisation is organised by algorithm 2. When there are more simultaneous events than a single thread should

handle, `Parallel` class sub-divide the list of events based on its size in equal parts and pass them on to further threads.

Algorithm 2 Mechanism of Parallel class

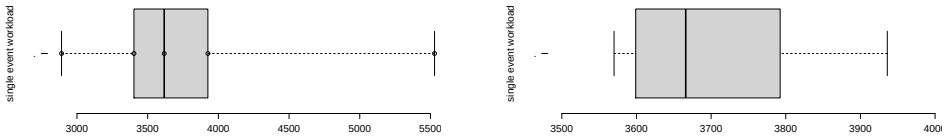
```
1: Procedure Parallel( list.lowIndex, list.upperIndex )
2: lowerIndex = list.lowIndex
3: upperIndex = list.upperIndex
4: Funct compute ()
5: if upperIndex - lowerIndex ⩽ threshold then
6:     while list.notEmpty do
7:         Execute events of list
8:     end while
9: else
10:    midIndex = (lowerIndex + upperIndex / 2)
11:    execute all (Parallel(lowIndex, midIndex), Parallel(midIndex,
   upperIndex))
12: end if
```

3.3.2 Evaluation

We have designed several scenarios to test the performance of the parallel version by focusing on time management while ensuring complete control over event occurrence. We also exploited our parallel version to foster the execution of our serverless environment.

3.3.3 Validation of the parallel event system

To ensure that the behaviour of our evaluation is following real life simulation patterns, we have instrumented the `JobDispatchingDemo` class of the dissect-cf-examples project. This class was already validated before to produce realistic simulations e.g., comparable to CloudSim. Our instrumentation focused on how the realistic simulation utilises the lowest abstraction layer of DISSECT-CF. We measured, the degree of parallelism, the typical event behaviour, the number of events in total and the average execution time of a single tick method call in nanoseconds (i.e., the single event workload) as shown in Figure 4a. To enable the comparison, we have also instrumented our parallel event system in the same way allowing us to acquire the typical workload of our synthetic tick methods.



(a) The execution time(ns) of single event workload using JobDispatchingDemo class

(b) The execution time(ns) of single event workload using our experiments classes.

Figure 4: Boxplot diagrams for JobDispatchingDemo class and our classes

Figure 4b shows the behaviour of our best approximate synthetic workload. Our median duration is within 3% of the realistic. The distribution of our workload is a bit narrower and more even, but the upper and lower whiskers of our synthetic experiment are within the typical range of the realistic simulation's values. As a result, from this point onwards, we will refer to synthetic workloads set up that used as the *original* single event workload.

3.3.4 Performance of the parallel event system

Our evaluation scenarios create 35,000 recurrent event objects. The object count was set so the minimum execution time of the sequential version is at least 5 minutes, allowing sufficient time for the parallelisation to take effect.

With the respect to the number of cores, there are two factors that influence the performance of the parallel version. First, the degree of parallelism (it denotes the number of events that happens at a specific time instance) plays a significant role to demonstrate the benefit of parallel version. We evaluated both the parallel and sequential versions of the simulator with four different degrees of parallelism (25%, 50%, 75%, 100%).

In 25% of parallelism, the parallel version runs 1.72 faster than the sequential. When the degree reaches 50%, the ratio increased to 1.74. The parallel version executes simulations 1.84 faster than the sequential version when 75% of all subscribed events occur recurrently during a simulation time. Finally, the parallel version reaches 2.01 times faster than the sequential version when the degree of parallelism is 100%.

Now let's analyse the effect of the size of the single event workload. We tested both of the parallel and sequential versions with various single event workload sizes, commenced with threefold lower than the *original* one to show the behaviour of simulating very low single event workload. Then reaching to threefold higher than the *original* single event workload to demonstrate the advantage of parallel version.

When the single event workload is threefold lower than *original* one, the parallel version runs 1.33 faster than the sequential version. This ratio increases to 1.67 when the single event workload is two times lower than the *original* single event workload. The parallel version even runs 2.11 faster than the sequential version using *original* single event workload. When the single event workload size doubled, the parallel version executes the simulation 2.32 times faster than the sequential version. The ratio increases to 2.44 when the single event workload size becomes threefold higher than the *original* one.

3.3.5 Evaluation of serverless environment using parallel event system

We exploited our parallel version to foster scenarios that happen in our serverless environment, particularly the **Cost Modeling and statistic** layer that mainly depends on the event system of DISSECT-CF. Our parallel version can be advantageous to these scenarios that require high performance, such as revealing the internal behaviour of provisioning resources and estimating the cost in our introduced environment. To demonstrate the performance of our parallel event system, we have conducted experiments on different workloads that require faster processing. We have generated workloads with different numbers of functions' invocations, namely, light-workload (100 thousands), normal-workload (1 million), and heavy-workload (10 million) by using our generator that we introduced in Subsection 3.1.

We then imitated the AWS Lambda provider and we simulated these workloads by our serverless environment. After that, we then analyzed the internal behavior of each simulation session, one times using the sequential event system and other using parallel event system. As both sequential and parallel versions produce the same simulation results (except execution time), Table 7 lists the performance metrics extracted from simulation sessions for all different workloads.

When we come to the simulation time for these workloads, the workload size and the degree of parallelism have significant influence in demonstrating the benefit of using the parallel version. Table 8 shows the execution time of simulated light, normal, and heavy workloads (the results listed in Table 7) in our serverless environment using sequential and parallel versions. The results show that the parallel version runs 1.24 faster than the sequential version during analyzing and simulating light-workload. This ratio increased to 2.24 when the workload was normal. Finally, the performance of the parallel version reaches 2.66 in heavy-workload compared to the sequential version. The results show that the parallel version demonstrates better performance for scenarios that engage heavy workload.

Table 7: Extracted performance metrics of different workloads using our serverless environment

Performance metrics	Light	Normal	Heavy
Cold-start probability (%)	2.542	0.385	0.135
Warm-start probability (%)	97.458	99.614	99.864
Arrival rate (s)	0.031	0.115	0.442
Average execution time of function (ms)	512	476	626
Number of unique functions	1256	1639	4660
Estimated cost ($)	2.503	23.264	305
Average Number of Concurrent instances	389	770	3913
Average lifetime instance (s)	1218	2199	6671
Average running-time instance (s)	126	947	3882
Average idle-time (s)	1092	1251	2789

Table 8: The execution time (s) of our serverless environment using sequential and parallel versions

Version	Light-workload	Normal-workload	Heavy-workload
Sequential	56	850	23204
Parallel	45	378	8703

Thesis 3 **Related Publications: [1, 4]**

I proposed a new parallel event system to foster the execution of DISSECT-CF-FaaS towards simulating large-scale scenarios. The introduced parallel version increases resource utilisation capability by allocating all available cores for backing the cost modelling and statistics of our serverless environment. The advantage of the parallel version is demonstrated when the simulated workload involves a large number of simultaneous events.

4 Summary

Simulators play a crucial role in the computing field by providing a flexible environment that could mimic real providers in the research area. As serverless computing is in its infancy, the research community needs to explore this promising cloud paradigm in a simulation environment to evaluate FaaS scenarios, and explore potential on architectures, operations, and mechanisms that could foster this computing paradigm.

In this dissertation, we proposed the DISSECT-CF-FaaS serverless environment. This integrated environment is capable of generating realistic traces that closely matches the original dataset's characteristics in terms of execution time, memory utilisation as well as user participation percentage. The evaluation in Subsection 3.1 showed that our generator approach provided excellent values for predicting generated trace attributes and users' invocations compared with the behaviour in the real-life dataset.

Our serverless environment is able to mimic the provisioning of resources, services, while also mimic the policy of the most well-known serverless providers. It also reveals the internal mechanisms and behaviours of the imitated providers by extracting performance metrics during simulation sessions. In Subsection 3.2, our evaluation showed that our environment provided the expected experimental results by, first, estimating the costs for various memory configurations. Second, reflecting provisioning policy properly to reduce cold-start. Third, capturing the behaviour of the trigger successfully. Finally, extracting average concurrent instances, running-time, and idle-time of involved instances from the simulation session.

Our DISSECT-CF-FaaS offers parallel execution to foster the scenarios that require high performance in computing, such as revealing the internal behaviour of the simulation session by extracting performance metrics. The evaluation of the parallel version in Subsection 3.3 showed that the execution of the simulation session can be sped up by 2.66 times compared to the sequential version. Thus, DISSECT-CF-FaaS is able to meet the expectations of the research community towards experiment various FaaS workloads and scenarios in a versatile environment.

5 Future works

We have identified three future research directions, namely, first we have to investigate other simulators that could use different trace formats, and exploit our generator to shift the generated realistic trace to these formats. To demonstrate to which extent our introduced approach can support these formats as well as to introduce modifications to adapt with them.

Second we hope to introduce other triggers such as the http trigger that enable DISSECT-CF-FaaS to interact and communicate with real applications to support other serverless scenarios such as dependent tasks in microservices applications. Finally we aim at focusing on the simulator's second most heavily used component in DISSECT-CF: the unified resource-sharing subsystem. This subsystem has high compute complexity, and its parallelisation will enable our model to rapid estimation of resource sharing on even larger scale-distributed systems. Applying these will lead to the seamless transition of the entire DISSECT-CF-FaaS into simulating billions of service invocations and their interactions in serverless computing situations.

Author's Publication and Software availability

[1] Sallo, D. H., & Kecskemeti, G. (2020). Parallel Simulation for The Event System of DISSECT-CF. In the 12th Conference of PhD Students in Computer Science: Volume of short papers Szeged, Hungary, University of Szeged, pages 58-61.

[2] Sallo, D. H., & Kecskemeti, G. (2020). Towards a DISSECT-CF extension for simulating Function-as-a-Service. In the 16th MIKLÓS IVÁNYI INTERNATIONAL PHD and DLA SYMPOSIUM abstract book, Pécs, Hungary : Pollack Press, ISBN: 9789634295785.

[3] Sallo, D. H., & Kecskemeti, G. (2021). Introducing Serverless Computing Model Based on DISSECT-CF Simulator. In the XXIV. Spring Wind Conference abstract book, Association of Hungarian PHD and DLA Students. Miskolc

[4] Sallo, D. H., & Kecskemeti, G. (2021). A Parallel Event System for Large-Scale Cloud Simulations in DISSECT-CF. Acta Cybernetica, 25(2), 469-484. **Scopus Indexed [Q3]**.

[5] Sallo, D. H., & Kecskemeti, G. (2022, June). Towards Generating Realistic Trace for Simulating Functions-as-a-Service. In Euro-Par 2021: Parallel Processing Workshops: Euro-Par 2021 International Workshops, Lisbon, Portugal, August 30-31, 2021, Revised Selected Papers (pp. 428-439). Cham: Springer International Publishing. **Scopus Indexed [Q2]**.

[6] Sallo, D. H., & Kecskemeti, G. (2023). Enriching computing simulators by generating realistic serverless traces. Journal of Cloud Computing, 12(1), 1-13. **Scopus Indexed [Q1]**.

[7] Sallo, D. H., & Kecskemeti, G. (2023). Towards a DISSECT-CF extension for simulating function-as-a-service. International Journal of Parallel, Emergent and Distributed Systems, 1-13. **Scopus Indexed [Q3]**.

The source code of this project is open and available (under the licensing terms of the GNU Lesser General Public License 3) at the following website:
Serverless trace generator: https://github.com/dilshadsallo/DistSysJavaHelpers
Serverless environment: https://github.com/dilshadsallo/dissect-cf-examples

REFERENCES

[BM02] Rajkumar Buyya and Manzur Murshed. Gridsim: A toolkit for the modeling and simulation of distributed resource management and scheduling for grid computing. *Concurrency and computation: practice and experience*, 14(13-15):1175–1220, 2002.

[CRB+11] Rodrigo N Calheiros, Rajiv Ranjan, Anton Beloglazov, César AF De Rose, and Rajkumar Buyya. Cloudsim: a toolkit for modeling and simulation of cloud computing environments and evaluation of resource provisioning algorithms. *Software: Practice and experience*, 41(1):23–50, 2011.

[JCSY19] Hongseok Jeon, Chunglae Cho, Seungjae Shin, and Seunghyun Yoon. A cloudsim-extension for simulating distributed functions-as-a-service. In *2019 20th International Conference on parallel and distributed computing, applications and technologies (PDCAT)*, pages 386–391. IEEE, 2019.

[Jou20] Soufiane Jounaid. *OpenDC Serverless: Design, Implementation and Evaluation of a FaaS Platform Simulator*. PhD thesis, Ph. D. Thesis, Vrije Universiteit Amsterdam, 2020.

[Kec15] Gabor Kecskemeti. Dissect-cf: a simulator to foster energy-aware scheduling in infrastructure clouds. *Simulation Modelling Practice and Theory*, 58:188–218, 2015.

[MK21] Nima Mahmoudi and Hamzeh Khazaei. Simfaas: A performance simulator for serverless computing platforms. *arXiv preprint arXiv:2102.08904*, 2021.

[RSBT19] Annanda Rath, Bojan Spasic, Nick Boucart, and Philippe Thiran. Security pattern for cloud saas: From system and data security to privacy case study in aws and azure. *Computers*, 8(2):34, 2019.

[SFG+20] Mohammad Shahrad, Rodrigo Fonseca, Íñigo Goiri, Gohar Chaudhry, Paul Batum, Jason Cooke, Eduardo Laureano, Colby Tresness, Mark Russinovich, and Ricardo Bianchini. Serverless in the wild: Characterizing and optimizing the serverless workload at a large cloud provider. In *2020 {USENIX} Annual Technical Conference ({USENIX}{ATC} 20)*, pages 205–218, 2020.