Sallo and Kecskemeti
Journal of Cloud Computing (2023) 12:36
https://doi.org/10.1186/s13677-023-00397-8

Journal of Cloud Computing:
Advances, Systems and Applications

RESEARCH

Open Access

Enriching computing simulators by generating realistic serverless traces

Dilshad Hassan Sallo[1,2*] and Gabor Kecskemeti[1,3]

Abstract

Serverless computing is stepping forward to provide a cloud environment that mainly focuses on managing infrastructure, resources and configurations on the behalf of a user. Research in this field can't rely on commercial providers such as AWS and Azure, as their inflexibility and cost often limits the required levels of reproducibility and scalability. Therefore, simulators have been opted as an alternative solution by the research community. They offer a reduced-cost and easy-setup environment. To get respectable precision, simulators use real traces collected and offered by commercial providers. These traces represent comprehensive information of executed tasks that reflect user behaviour. Due to serverless computing's recency, typical workload traces employed by IaaS simulators are not well adoptable to the new computing model.

In this paper, we propose an approach for generating realistic serverless traces. We enhance our previous generator approach that was based on the Azure Functions dataset. Our new, genetic algorithm based approach improves the statistical properties of the generated traces. We also enabled arbitrary scaling of the workload, while maintaining real users' behaviour. These advances further support reproducibility in the serverless research community. We validated the results of our generator approach using the coefficient of determination (R^2), which shows that our generated workload closely matches the original dataset's characteristics in terms of execution time, memory utilisation as well as user participation percentage. To demonstrate the benefits of the reusability of the generated traces, we applied them with a diverse set of simulators and shown that they offer reproducible results independently of the simulator used.

Keywords Serverless workload, Serverless trace, FaaS, Genetic Algorithm

Introduction

Serverless is a new computing paradigm adopted by several cloud providers. It provides a new style service, letting a user to mainly concentrate on coding rather than managing infrastructure and operations [1]. However, in most cases, real providers are not always the most favourable choices for researchers to execute and evaluate their desired scenarios, due to having the costly and complex environment (leading to non-reproducible results) [2]. One alternative solution is simulators, which were opted by the research communities to evaluate scenarios in reduced-cost and easy-to-setup-environment [3]. To simulate serverless functions, realistic workload needs to be used during the research scenario evaluation process. Unfortunately, workload traces typically employed by IaaS simulators are not well adoptable for the serverless model, thus they do not represent the new kind of user behaviour. This highlights the need to enrich simulators by versatile traces representing realistic serverless workloads.

*Correspondence:
Dilshad Hassan Sallo
sallo@iit.uni-miskolc.hu
[1] Institute of Information Technology, University of Miskolc, Miskolc, Hungary
[2] Department of Computer Science, College of Science, University of Duhok, Duhok, Iraq
[3] Department of Computer Science, Liverpool John Moores University, Liverpool, UK

Springer Open

© The Author(s) 2023. **Open Access** This article is licensed under a Creative Commons Attribution 4.0 International License, which permits use, sharing, adaptation, distribution and reproduction in any medium or format, as long as you give appropriate credit to the original author(s) and the source, provide a link to the Creative Commons licence, and indicate if changes were made. The images or other third party material in this article are included in the article's Creative Commons licence, unless indicated otherwise in a credit line to the material. If material is not included in the article's Creative Commons licence and your intended use is not permitted by statutory regulation or exceeds the permitted use, you will need to obtain permission directly from the copyright holder. To view a copy of this licence, visit http://creativecommons.org/licenses/by/4.0/.

In our previous work [4], we proposed an approach to generate realistic traces from the Azure Functions dataset. This approach was integrated with the DISSECT-CF simulator. Our past mechanism was based on the dataset's invocation level execution time and memory utilisation percentiles. For all FaaS's invocations, it generated execution time and memory utilisation to reflect the consumed memory and running period while it was executed in a serverless platform. 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.

This paper's main contributions address these previously existing drawbacks by applying the following methodology. First, we improved the previous approach by using a genetic algorithm that adjusted our generator's inputs to better match the dataset's original percentile values and task distributions. The introduced technique holds all the properties of the previous approach, but it better resembles the behaviour observable in the Azure Functions dataset. Second, we introduced the capability to scale the workload to fit any desired scenario and infrastructure while maintaining the real user behaviour disclosed in the dataset. Moreover, we enabled researchers to customise and generate a workload that closely matches particular details about the dataset (like only model a particular time period or particular trigger action for the functions). Finally, we have also provided the option to generate traces to widely used formats. This enhances the reusability and reproducibility of research based on the generated traces. It also eliminates the need to repeatedly apply the genetic algorithm based adjustment of the trace generator during each experiment. Finally, this also provides an alternative way to simulate serverless behaviour with IaaS simulators that do not natively support the serverless model.

To demonstrate the usability of our approach, first, we generated several traces involved functions with arbitrary invocation numbers. They also include different scaling workloads to observe users' behaviour. Second, we validated our generated approach using R^2 of the reported execution time, memory utilisation, users' behaviour and percentiles between both the original Azure Functions dataset and our generated ones. We compared the generated standard traces of this approach with the production of the pervious approach that internally generates traces. Finally, in terms of simulation, we have simulated the generated standard format traces using cloud, grid and serverless frameworks.

To validate our generator approach of realistic function invocations with real behavior, we have compared the generated traces to the originals on the following way. First, we calculated the percentage of users' invocations that reflect their behaviour over one complete day. Second, we randomly selected different numbers of functions to generate the different size workloads. Third, we calculated the percentage of users' invocations and percentile values of execution time and memory utilisation for generated workloads. Finally, we compared the already percentiles and calculated the percentage of user's behaviour from the original trace with the generated ones with the help of (R^2). Our approach provided very good (R^2) (> 0.99) values for predicting percentiles and users' invocations with relatively well matching behaviour to the originals found in the azure traces.

In terms of converting generated traces, we have validated the converted traces by simulating them using various simulators, which showed very good results.

Related work

The cloud research community has been using simulators as an alternative solution for the services provided by providers to evaluate algorithms and scenarios. Simulators offer high levels of freedom to configure scenarios and experiment in a low-cost environment. As a result, various distributed computing infrastructure simulators were built during the last decade with different functionalities and features such as CloudSim [5], DISSECT-CF [6] and GridSim [7].

To imitate the behaviour of computing infrastructure providers with respectable precision, simulators use real traces collected from providers. These traces represent users' behaviour by showing the type of executed task and consumed resources. Unfortunately, workload traces collected from past providers are not well adoptable for modelling infrastructure workloads caused by serverless frameworks.

Several serverless simulators have been developed recently. Using them for evaluating new approaches to manage FaaS systems could still lead to potentially misleading results. This is due to the unavailability of realistic, large-scale serverless traces in these simulators. DFaaSCloud [8] started addressing the missing serverless functionalities of CloudSim. They added basic FaaS components on top of the simulated distributed cloud architecture. However, this new simulator is unable to utilise large scale generated realistic serverless traces. Instead, it provides synthetically generated behaviours via classes that define events and requests based on predefined

function profiles. OpenDC Serverless [2] is developed to model and test custom FaaS patterns. They have developed a benchmark for evaluating performance of the particular providers rather than focusing on supporting the research communities, thus it is out of our paper's scope. FaaS-Sim [9] is a simulator that mainly focuses on supporting edge functions by extending existing platforms such as Kubernetes and OpenFaaS. It uses a container scheduling system to efficiently use edge resources and simplify the development of edge computing applications. However, FaaS-Sim uses traces collected from edge testbed and synthetic traces that do not contain all serverless functions attributes. SimFaaS [10] is a serverless framework developed to mainly focus on predicting the performance metrics of a workload. SimFaaS also allows creating and customising synthetic workloads. Also, it can read traces of their proprietary format. This format was derived from their interactions with amazon's lambda and web services framework.

In our previous approach [4], we aimed at generating realistic traces resembling the behaviour observable in the Azure Functions datasets. The generated traces were shown to have the same average execution times and memory utilisation values as the original dataset has shown. However, any further statistical comparison between the generated traces and the dataset lead to issues. This makes the generated traces not suitable for a wide variety of research. Apart from this, our previous approach has suffered from the following issues: (*i*) it was strongly bound to DISSECT-CF, (*ii*) it was not capable of scaling the generated workload to fit the necessary research scenario. These issues were caused by the following behaviours: first, every time there was a need for a serverless trace, we have had to go through the process of generating the traces from the start. Second, other simulators were unable to simulate with the generated traces as they were not adapted to this approach.

Based on the above points, there is a clear gap in realistic, large scale trace generation for serverless platforms. Thus, this paper aims at presenting a realistic quality, scalable trace generator that reflects the user behaviour and contributed tasks.

Methodology

Our previous generator approach

Microsoft Azure Functions released a dataset representing 14 days of serverless function execution history [11]. The dataset offers per minute details of per trigger (e.g., http, timer) function invocation counts. Alongside these, it also characterises the runtime and memory utilisation characteristics of a particular function invocation type. This characterisation is done in terms of disclosing the distribution of the utilisation via a few percentile values (i.e., they specified what was the runtime/memory value for 0^{th}, 1^{st}, first quartile, median, third quartile, 99^{th} and the 100^{th} percentile). The percentiles were provided on a daily basis showing the distribution execution time and memory utilisation for invocation count that were binned at 1-minute intervals. Finally, they not only offered the median, also the minimum, maximum and average values for each function invocation kind's runtime and memory use.

Our previous approach [4] aimed at generating realistic traces based on this dataset. This past approach allowed a user (who requires to produce trace) to select a particular function invocation kind and time period of a particular day to act as the model to generate traces. During the generation, the approach reused the submitter and function related information in the generated trace, while it provided new values for not-explicitly disclosed values like arrival time, runtime and memory utilisation. To generate these, we used the disclosed percentiles and minimum and maximum values. We validated this approach by generating several traces containing functions that were selected randomly from the complete dataset (i.e., we asked the generator to try to mimic the behaviour of a particular function on a particular day during a simulation). We then collected the average runtime and memory utilisation values and shown that the coefficient of determination (R^2) was > 0.99. This suggests that we have produced an approach that is providing realistic average runtime and memory utilisation values. We also calculated the percentiles for the generated trace and the percentile values were having significantly weaker R^2 values due to the trace's way of disclosing the percentiles (i.e., they did not disclose the actual percentiles but they first calculated average runtime and memory utilisation values over 30 second periods throughout the day, then this was used as the basis of the percentile calculation).

Dataset that is collected from the Azure provider reflect users' behaviour (a user has unique HashOwner in dataset) by showing the number of invocations (tasks), type of service and how frequently particular tasks are invoked. Enclosing the real information is crucial for further studies such as predicting the consumption behaviour of users to stimulate resource usage awareness. The Azure dataset contains a huge number of tasks and services, which could not meet several researchers' scenarios that require small-scale workload with real users' behaviour. Unfortunately, this previous approach did not consider this feature to support this scenario. It also neglects generating workloads based on triggers that came with original dataset such as http, timer, event, queue, storage, orchestration and others.

One of the observed limitations of the past approach is the lack of workload reusability. As it is integrated

internally with a serverless model of DISSECT-CF simulator, every time there is a demand of serverless traces, we must go through the process of generating realistic traces completely. Moreover, other simulators have no opportunity to advantage from the generated realistic serverless traces as they are not adapted to this approach.

Proposed architecture

To remedy the aforementioned limitations, we updated the architecture of the previous approach (see Fig. 1) by adding three further components, namely `GA`, `User Behaviour` and `Standard Format`. They are added to improve quality, enabling scaling workload and providing reusability of generated traces respectively. Our new extension also relies on the `DistSysJavaHelpers` project that provides abstractions to represent arbitrary workloads as well as enabling loading of several well-known workload trace formats.

Our extended architecture consists of three components compatible with the previous generator approach. These components are distributed over the following two layers. **Generation and setup layer**, which is responsible for generating traces based on a selected dataset. **Processing layer**, which is responsible for detecting the users' behaviour and converting the generated traces to standard formats.

Here we will give an overview about introduced components and in forthcoming subsections, we will explain them in details. The `GA` component includes the concept of genetic algorithm, and it works with other components of the same layer to improve the process of generating traces based on averages and percentiles. The `User Behaviour` component calculates the number of invocations and services for each unique user to provide statistical information allowing scale-workloads. Finally, the `Standard Format` component supports the reusability of the generated traces by converting them to other formats that are supported by most simulators.

Improve the percentiles of generated trace

The previous approach generates one time a set of values (representing one individual in genetic algorithm concept) for each unique function and its invocations. The production of our approach was based on the percentiles of execution time and memory utilisation, which determine how the values will be generated and distributed. As a result, we will generate one time the set of execution time and memory utilisation values to represent data of function's invocations. Although our approach generates values within the range of percentiles, good values of (R^2) could not be produced by a single iteration as they generated randomly.

To address this issue, we introduced a genetic algorithm (`GA` component) to our previous approach to produce several sets of values, and then it selects optimal values to constitute a single set for function and its invocations as shown in Fig. 2. Our approach is able to generate full large-scale traces or scaling-workload to desired size (which will be discussed in the following subsection). When the Azure dataset is selected, it will be processed by `Generic Trace Producer` to analyse dataset's contents. It reads each line from the trace file (which represent one unique function and its invocations over a 24 hour period) and imitates the behaviour of the function according to invocations, as shown in Fig. 2. This is done by extracting the necessary percentile information to be passed for `Generate Execution Time` and `Generate Memory` components via `GA` component. The `GA` component includes the configuration of genetic algorithm set according to [12], in terms of generation's number, individuals and fitness value that has to be fulfilled.

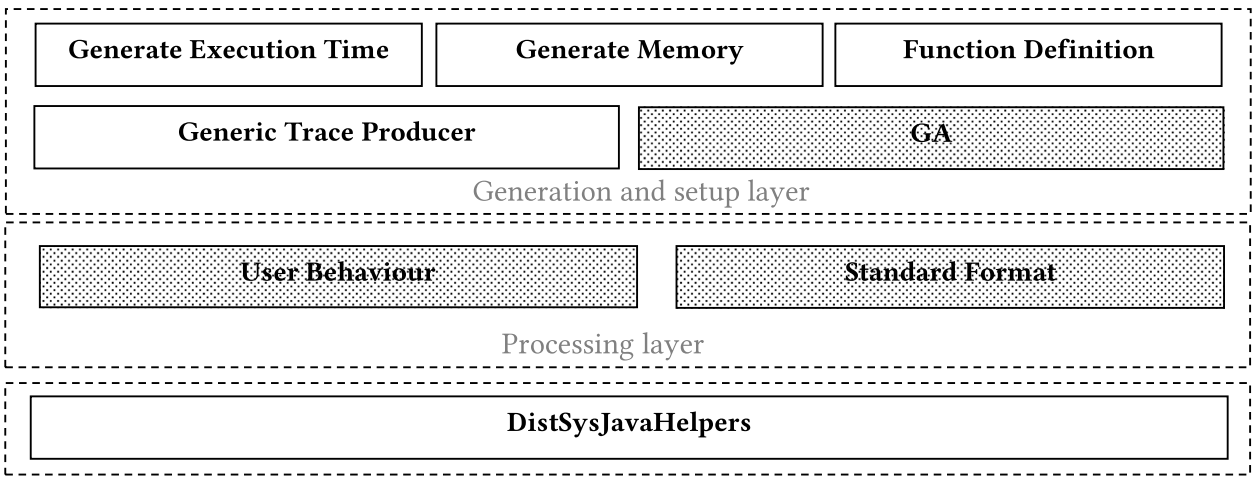

Fig. 1 Architecture of proposed model

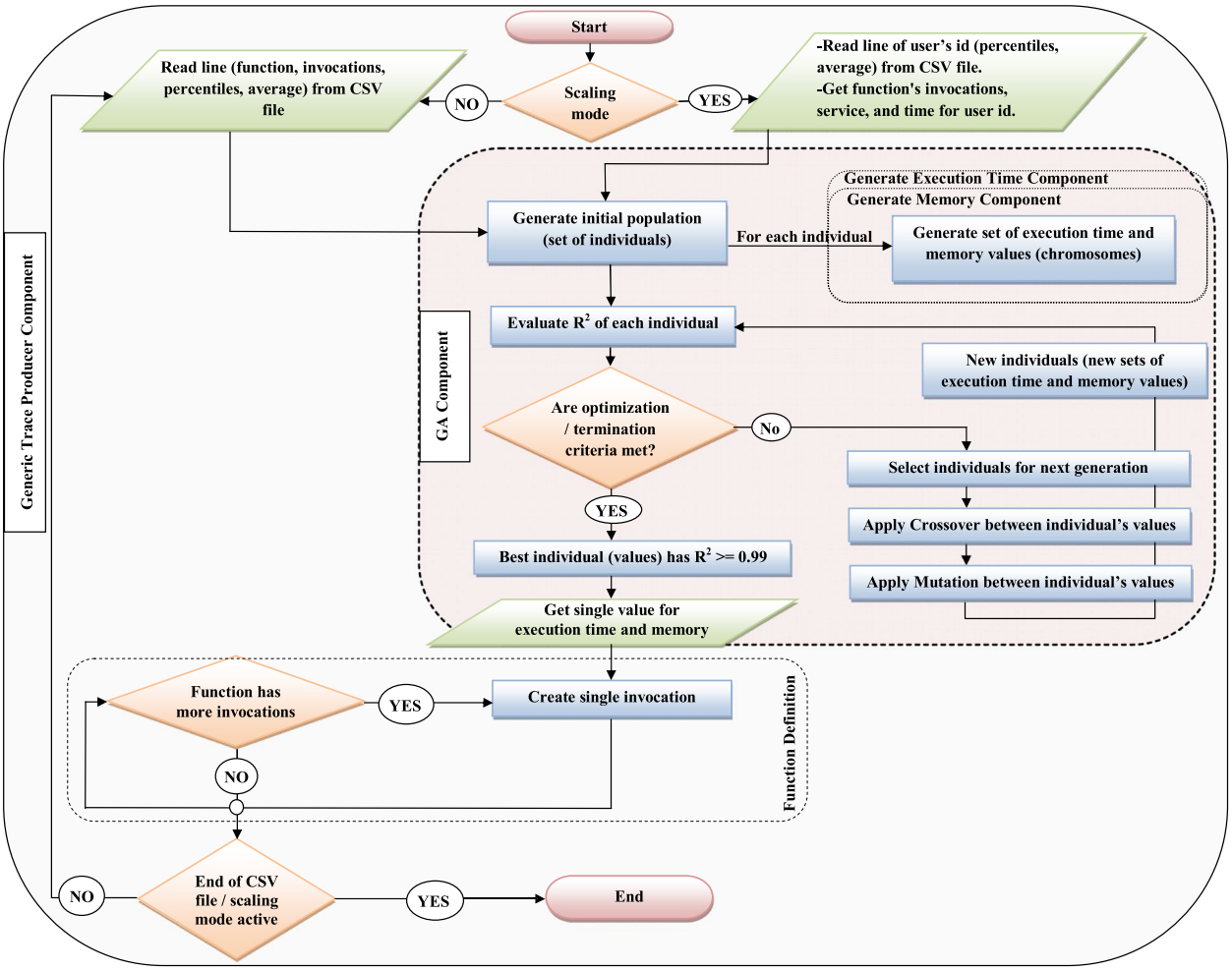

Fig. 2 Flowchart for process of generating trace

For each unique function and its invocations, the `GA` component produces sets of values according to the number of individuals. Each set of these represents generated execution time and memory utilisation values for the selected function and its invocations. For each iteration, the `GA` component will apply the selection, mutation and crossover policies, according to the configuration, to select the best amongst the whole selection of individuals that produce good R^2 value for the percentiles (fitness function) against the original percentiles of the Azure dataset.

The process of iteration will continue, according to the selected number of generations, which finally will conclude with good execution time and memory utilisation values for the selected function and its invocations.

After that, `Function Definition` component (which is responsible for populating the corresponding task definition in the simulation) will instantiate invocation with the previously extracted values (e.g., amount of memory). Each single invocation involves several parameters such as unique ID, submitted time, execution time and memory utilisation, that determine the behaviour of the function. `Function Definition` will continue the generation of the tasks of the selected function according to its total number of invocations. After generating all the required function invocations for the simulator, `Generic Trace Producer` proceeds reading the next line from the trace. This process continues till we finish generating all requested functions and their invocations.

Scaling workload with real users' behaviour

Scaling-workload is essential to support researchers' scenarios that require small infrastructure to achieve their desired purposes. 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. The main logic behind this component is explained in Algorithm 1.

Algorithm 1 Scaling workload with users' behaviour

```
1: procedure ScalingWorkload(dataset, workloadSize, service, time)
2:     totalTasks ← Σ_{i=0}^{N} nt_i
3:     workload ← ∅
4:     for id_{u_x} ∈ ID do
5:         involvedPerc ← ut(id_{u_x})/totalTasks
6:         scaledTasks ← ⌊involvedPerc · workloadSize⌋
7:         workload ← workload ∪ genericTraceProducer(id_{u_x}, scaledTasks, service, time)
8:     end for
9: return workload
10: end procedure
```

Before we introduce the algorithm, we discuss its foundational notation. It is based on the *dataset* that is read in the beginning to identify each user and its behaviour. The complete dataset of N lines is represented as: $dataset = \{line_1, line_2, \ldots line_N\}$. During scaling, we model each line as a tuple: $line_i = (id_i, nt_i)$, where id_i depicts the hash owner identified by the i^{th} line, and nt_i specifies the number of tasks listed in the particular line. User hashes can repeat over several lines in the trace when the same user returns and uses the infrastructure for multiple functions. Thus, it could happen that $id_i = id_j = id_{u_x}$. To ensure correct user representation in our scaling approach, we collect the unique hashes (depicted as id_{u_x}) in the set of $ID = \{id_{u_1}, id_{u_2}, \ldots id_{u_K}\}$. During the loading of the dataset, we also determine each individual user's overall workload in the dataset. We use the following notation for this calculation: $ut : ID \rightarrow \mathbb{N}$, and we calculate the value of this function as follows: $ut(id_{u_x}) = \sum_{i:id_i=id_{u_x}} nt_i$, where id_{u_x} is a unique hash that we previously collected in the set ID.

Based on these definitions, our algorithm, first collects the total number of tasks in the complete dataset - see line 2. This goes through all the lines and sums up their individual task counts. Next, the algorithm also starts by initialising the generated *workload* completely empty. This allows us to generate individual sub-sets representing particular users from the original dataset and then merge them together in the next phase.

After the preparatory steps, our algorithm goes through all unique user ids found in the original trace and generates a sub-scaled workload for each of them. This is done by first calculating the involvement percentage of each user based on its total number of tasks (*ut*) and the total number of tasks in dataset (see line 5). After that, we calculate the expected number of invocations in the scaled workload for each user based on their participation percentage and the expected *workloadSize* (see line 6). Note, that we round the result down to the nearest integer ensuring that our scaled trace only contains user ids that would result in at least one generated invocation. Finally, in line 7, we generate the required number of invocations following our previously discussed `Generic Trace Producer` component. This, then produces a single user focused statistically correct generated set of invocations that fits the desired scenario of the user of our scaler. These, unique-user-specific traces are then merged into the finally returned generated workload. To allow further customisation of the scaled traces, we enabled customising the scaled workload based on a particular time range or specific services. These are both passed in as the *service* and *time* parameters to both the scaler as well as the trace producer.

Converting generated trace to standard formats

`Standard Format` component enables reusability of generated traces by converting them to other formats and getting rid of the process of generating traces repeatedly. When the tasks (FaaS functions) are generated, this component obtains each task with all its attributes (e.g. execution time, tasks' id and amount of memory) to store in its repository. Then, for each task, it sorts its attributes values based on the desired format. Finally, it goes through the process of extracting the tasks one by one till the end.

`Standard Format` component supports converting generated traces to several standard formats. This provides an alternative solution to other simulators that don't natively support the serverless model to advantage from the generated realistic serverless traces. It also allows a user of the model to write its own format by extracting all attributes of generated tasks based on the desired format.

`Standard Format` component enriches serverless frameworks with FaaS workload by converting generated traces to AWS Lambda traces and other trace formats. However, some AWS trace attributes are not available

in Azure datasets, such as cold start, which demonstrate when a provider has requested a new instance for an invocation. To collect these attributes in real run time, our serverless model [4] that was devised by extending the DISSECT-CF, uses converted AWS Lambda trace as input for simulation to produce the necessary attributes that are unavailable. For each task (invocation) in trace, our model will submit the task to the infrastructure to be simulated. If there is an instance ready to accommodate this task, it will simulate directly. Otherwise, the model will request a new instance (cold start) for this task. When the simulation is finished, the model reproduces the AWS trace with complete attributes to be simulated by other serverless frameworks.

Evaluation

A laptop (Intel (R) Core (TM) i7-4600U CPU @ 2.10GHz (4 CPUs), 2.7GHz, 8 GB) was used for the evaluation of our approach and model for generating and simulating FaaS.

Percentiles

To validate our approach of generating realistic traces with the help of the genetic algorithm, we configured the `GA` component setting according to an analysis provided in [12]. We used 100 individuals and tournament selection with size equal to 10 individuals. Regarding crossover and mutation rates, they set to 0.9 and 0.05, respectively. Finally, the elitism strategy is exploited to copy the best individual from the current population into the next one without undergoing the genetic operators.

After that, we used the first day of the Azure dataset that contains around 36000 unique functions. We then 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. As a result, large-scale of execution time and memory utilisation values were generated. For each function, we calculated those percentiles from generated 5000 invocations to see how close they are to the original percentiles of the Azure dataset. We used the coefficient of determination (R^2) between generated and original percentiles to show data accuracy.

Although the selected functions produced wider range of values for the model and observed data compared to result in previous approach as they have chosen randomly (each has different distribution values), we concluded with a very good result of R^2, which was 0.9994 for execution time and 0.9995 for memory utilisation as shown in Fig. 3. This indicates the accuracy (and realism) of our approach and how genetic algorithm has enhanced the generated percentiles that affected the result of the previous generator approach.

Users' behaviour

To validate users' behaviour, we have chosen the second day of dataset, which contains a comprehensive history of users, functions and services that executed in the real 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, as shown in the Table 1.

After obtaining all information, they will be passed to `Generic Trace Producer` component to be used for producing different workloads with the same Azure dataset users' behaviour. We validated the scaling approach by producing different workload sizes that fit small and large infrastructures. 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 one (which existed in selected file representing one day) by using R^2 as shown in Table 2. 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 that the generated execution time and memory utilisation percentiles resemble the original values during scaling workloads.

The `Generic Trace Producer` also enables a user of a simulator to generate workload with customized options such as a specific service or time while maintaining real users' behaviour with help of `User Behavior` component. We generated a trace as shown in Fig. 4 for orchestration trigger that contains numerous invocations during one day. We also generated a trace for the timer trigger, and we showed the participation percentage of users at the first and last minutes of the day, as shown in Fig. 5.

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 frameworks that belong to different fields, namely, DISSECT-CF (cloud simulator), GridSim (Grid simulator) and simFaaS (serverless simulator).

DISSECT-CF

One of the approaches to verify the reliability of converting generated traces is monitoring the internal behaviour

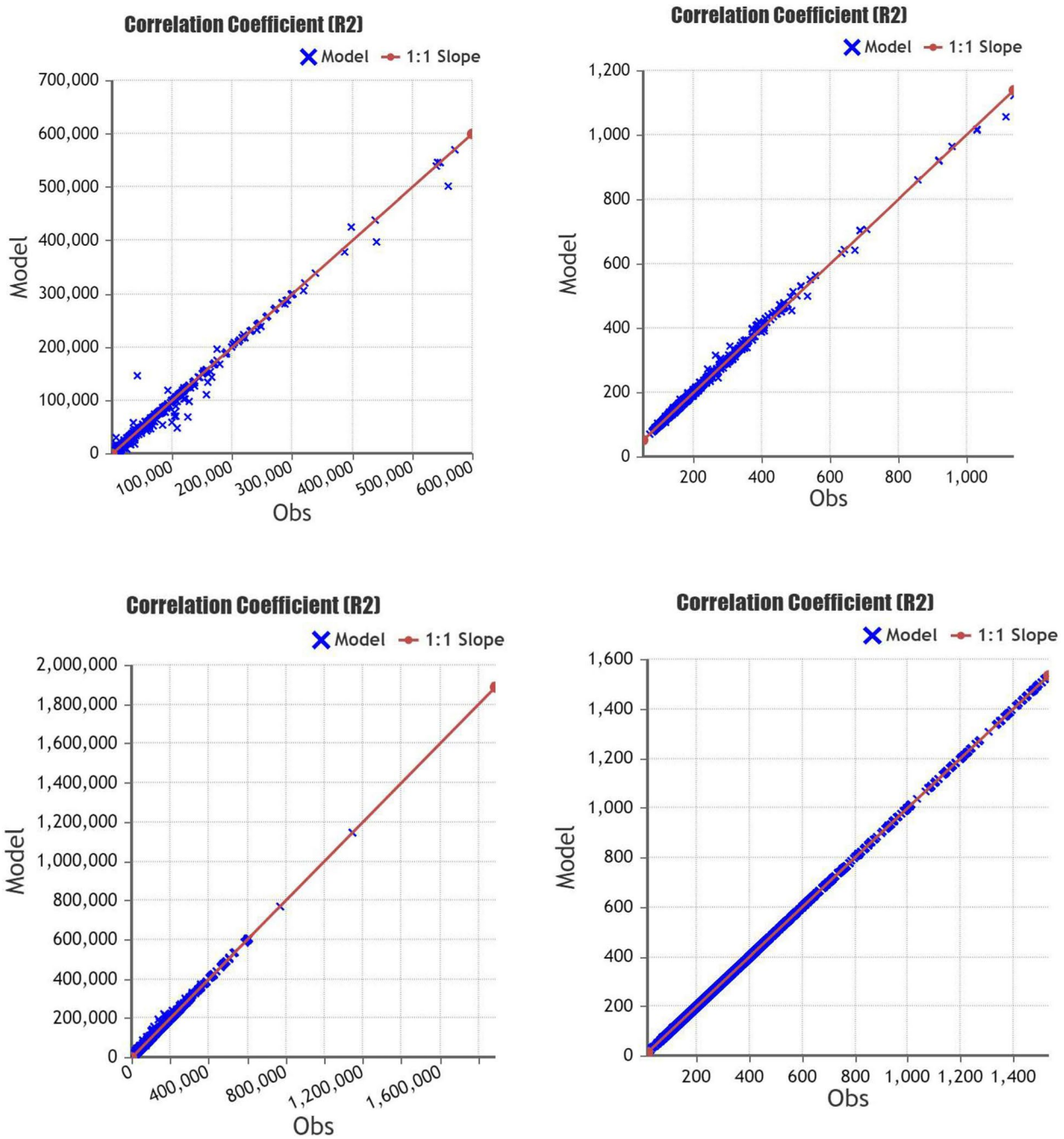

Fig. 3 Evaluation of percentiles for execution time (left-figures) and memory utilisation (right-figures) using Coefficient of determination: the result of previous generator approach (top 2 plots), and proposed approach using GA (bottom 2 plots)

of the simulation while simulating the same tasks in 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 precision results. DISSECT-CF will be used as a standard to compare with other simulators in the forthcoming subsections.

To show the accuracy of converting traces to other formats, we first configured the simulated infrastructure of DISSECT-CF for experiments by setting up a homogeneous cloud with 100 physical machines (each configured with 32 CPU cores, 256 GiB of memory and 256GB of storage) and configured central data storage of 36 TB.

Table 1 Top 10 users by number of tasks submitted to the real provider

Rank	UserID	Number of Jobs	Percentage
1	U4932	127471686	14.94%
2	U376	98867247	11.59%
3	U5660	51372036	6.02%
4	U1746	49036404	5.75%
5	U3387	37036087	4.34%
6	U8104	22457567	2.63%
7	U6940	16743959	1.96%
8	U6143	15937341	1.87%
9	U2488	15808936	1.85%
10	U6175	14780372	1.73%
11	Other	403617380	47.31%
12	Total	853129015	100.00%

Table 2 Scaling workloads with real users' behaviour

Workload Size	R^2 (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

Second, we generated 20 thousand tasks (medium-scale trace that is suitable to above configuration) from the third original Azure trace (CSV file). We then asked to convert the same generated trace to Grid Workload Format (GWF) and Standard Workload Format (SWF) traces. As a result, the generated trace (CSV) was directly simulated by DISSECT-CF. In the second round, we simulated the GWF and SWF traces. Finally, in order to proof that two approaches are the same and achieve our expectation by having accepted percent difference value (5%), we have to observe the internal behaviour of the simulator in terms of simulated timespan (time that trace takes in simulated work not in real life), number of used virtual machines, average utilization of physical machines and total power consumption for all experiments.

Table 3 shows that the percent difference of simulation timespan and the number of used VMs are identical between the original trace (CSV) and converted traces (GWF and SWF). However, the results are 1.3% and 0.08% for the average utilisation of PMs and total power consumption, respectively. This meets our expectations and shows that our approach of converting is properly and realistic.

Other simulators

Enriching frameworks with serverless traces offers opportunity to simulate the behaviour of FaaSs, which in the future enabling conduct further experiments for investigating other features such as containerisation, energy consumption, monetary cost, that are out of our paper's scope. To validate our approach of converting traces and demonstrating their usability by other simulators, we explored the most popular open-source frameworks in the computing field, as shown in Table 4, 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

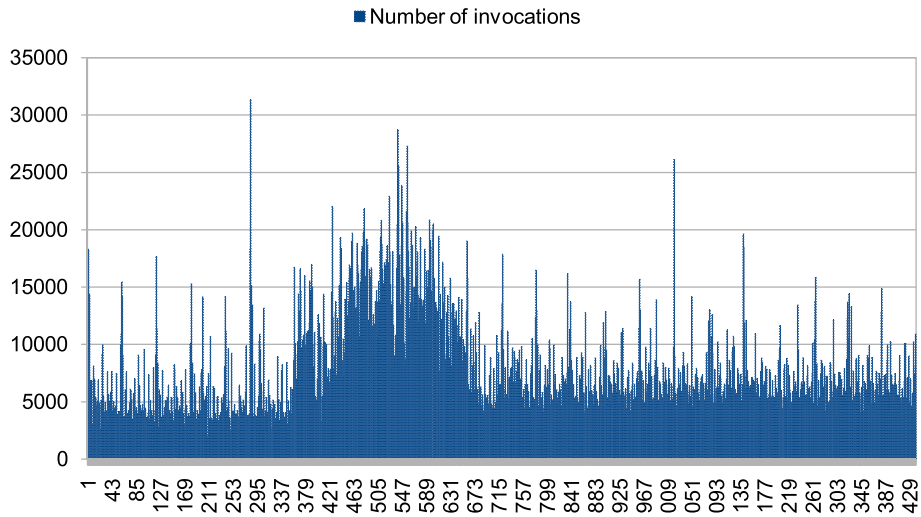

Fig. 4 Number of orchestration trigger invoked during one day

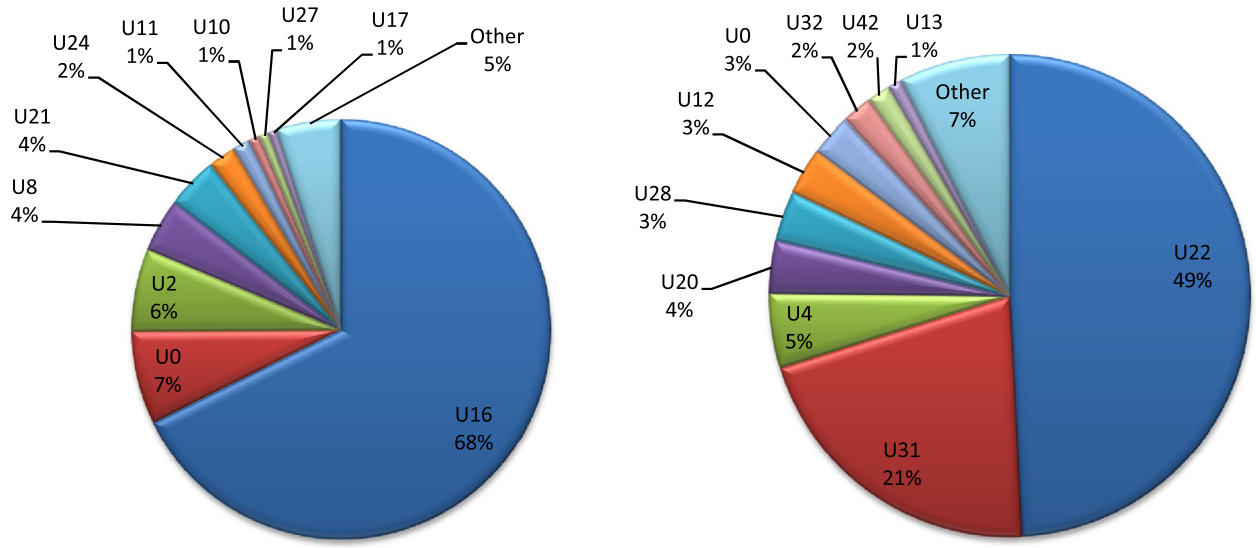

Fig. 5 Percentage of invocations for timer trigger per user, Left-figure: first minute of the day, Right-figure: Last minute of the day

Table 3 Simulating 20 k jobs 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

workloads are supported by these simulators. We have selected the most recent simulator used in the evaluation CloudSim [5] to validate our converted traces against the validated simulator DISSECT-CF. CloudSim is popular simulator and support both real traces and generates a synthetic workload for simulating. We used the same cloud configurations mentioned earlier for both DISSECT-CF and CloudSim. We generated traces in GWF and SWF formats originally for DISSECT-CF, and we then reused them for CloudSim. Unfortunately, the result was very different in terms of simulated timespan. CloudSim could not even be close to it. After investigating the factors that influenced the result, we have concluded that CloudSim does not consider the submitted time of a task (which represents the time of dispatch task to the real provider) and it dispatches all tasks at the same time in the beginning of the simulation.

We then have selected the second recent and popular simulator GridSim [7] for simulating and validating our converting approach. GridSim considers all attributes of a task, such as execution time, submit time, memory

Table 4 Open-source simulators used recently in evaluation

			Workload supported	
Simulator	Type	Year Used	Trace file	Synthetic
CloudSim [5]	Cloud	2022	✓	✓
GridSim [7]	Grid	2022	✓	✓
iCanCloud [13]	Cloud	2021	✓	✓
DISSECT-CF [6]	Cloud	2021	✓	✓
WorkflowSim [14]	Cloud	2021	✓	N/A
CloudSched [15]	Cloud	2021	✓	N/A
CloudAnalyst [16]	Cloud	2021	N/A	✓
GreenCloud [17]	Cloud	2021	✓	✓
GPUCloudSim [18]	Cloud	2021	N/A	✓
SimGrid [19]	Grid/Cloud	2021	✓	✓
DFaaSCloud [8]	Serverless	2021	N/A	✓
BigHouse [20]	Cloud	2021	✓	✓
simFaaS [10]	Serverless	2021	✓	✓
FaaSFSim [9]	Serverless	2021	✓	✓
CloudSimSDN [21]	Cloud	2021	✓	N/A
CEPSim [22]	Cloud	2021	N/A	✓
OpenDC Serverless [2]	Serverless	2020	✓	✓
FederatedCloud-Sim [23]	Cloud	2020	✓	N/A
EMUSIM [24]	Cloud	2019	N/A	✓
CloudReports [25]	Cloud	2019	N/A	✓
DCSim [26]	Cloud	2018	✓	N/A
ElasticSim [27]	Cloud	2017	✓	N/A
Cloud2Sim [28]	Cloud	2016	✓	✓

Table 5 Comparing simulating tasks 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%

utilisation and others, as the same as DISSECT-CF does. We have simulated different workload sizes starting from small-scale trace (1 thousand tasks) to large-scale trace (1 million tasks) using DISSECT-CF and GridSim. We then have calculated the percent difference of simulated timespan for each workload, as shown in Table 5. 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.

SimFaaS

Our approach supports converting generated traces to various serverless workload formats that used by serverless simulators. To demonstrate that, we have selected simFaaS to simulate converted traces as it supports real traces and it was recently used for real evaluation as shown in Table 4.

SimFaaS [10] is a serverless framework built to optimize the performance of FaaS applications by predicting several Quality of service (QoS) related metrics accurately. SimFaaS used real-world traces from Amazon AWS Lambda to conduct experiments and extract performance metrics from simulation. These metrics include calculating the probability of cold start, the average number of idle instances and average utilization of instances.

As our 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 our serverless model [4]. It then reproduces the AWS traces with complete attributes to be simulated by simFaaS framework. Our model also provides performance metrics likes simFaaS. These include calculating the arrival rate of requests and counting the idle instances by calculating the difference between the total number of instances and the number of instances that running at each second. Moreover, it measures the probability of cold start by dividing the number of requests causing a cold start by the total number of requests made during the simulation. Finally, it measures the average of instance utilisation by counting the number of unique instances in the warm pool at each second.

To check the accuracy of our model output, we first, simulated the real-world traces from Amazon AWS Lambda (enclosed with simFaaS) using simFaaS simulator and our model. These traces contain around 500 thousand tasks (invocations). As both simulators extract performance metrics from simulation, we measured the coefficient of determination (R^2) between the simFaaS and our model results to show data accuracy. For arrival rate, R^2 was 0.9999 and 0.9964 for cold start probability. R^2 were 0.9962 and 0.9982 for average utilisation and idle instances, respectively.

Second, we chose one of the Azure traces representing one day of the execution history. This trace contains around 36 thousand unique functions with their requests. We then generated 15 AWS Lambda traces from this Azure trace. For each trace, we have randomly selected thousand unique functions with their requests. As a result, the generated traces contain around 500 thousand requests. After that, we simulated these traces with our model to enclose the attributes that are not available, as well as to extract the performance metrics from the simulation. Finally, we simulated the AWS traces that reproduced by our model as output, using simFaaS framework.

We measured R^2 of the performance metrics extracted from the simulations as shown in Fig. 6, for both simFaaS and our model to validate the converting approach. 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 simFaaS serverless simulator.

Conclusion and future work

The simulators play a crucial role in the cloud computing, grid computing, and serverless computing fields by providing a flexible environment that could replace real providers in the research area. Imitating such serverless environments requires realistic traces that reflect users' behaviour in terms of execution history. Due to existing IaaS workloads are problematic and not well adaptable for use by serverless simulators, we proposed an approach to generate realistic traces based on the Azure functions dataset. Our approach produced realistic traces and supported the reusability of the generated traces by converting them to standard formats that adapted to various computing simulators. It also supported

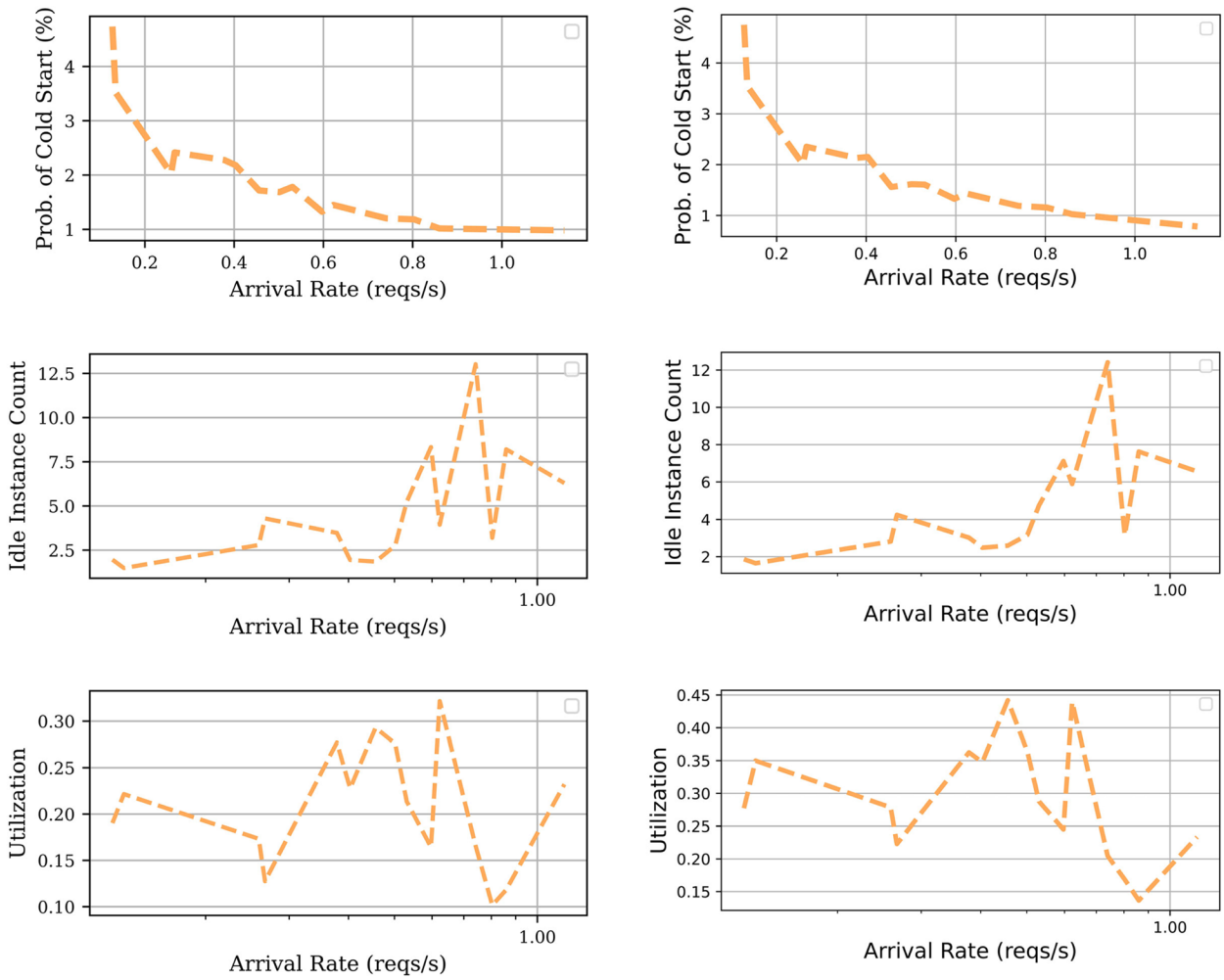

Fig. 6 Performance metrics extracted from simulation, Left-figures: simFaaS, Right-figures: our model

scaling workloads with real users' behaviour. The results of evaluation showed that generating the percentiles of the execution time and memory utilisation with help of the genetic algorithm resembles the Azure dataset. The scaling workloads from small-scale to large-scale trace showed a very good R^2. Finally, converting generated traces to standard formats and serverless workload demonstrated reusability by other simulators.

Future work will focus on investigating other simulators, which could use different trace formats. To demonstrate to what extent our introduced approach can support these formats as well as to introduce modifications to adapt with them.

Acknowledgements
Not applicable.

Authors' contributions
DHS carried out the approach of generating serverless traces including coding and drafted the manuscript; GK helped develop the concepts, reviewed, and revised the manuscript. Both authors read and approved the final manuscript...

Funding
Open access funding provided by University of Miskolc. This research was partially supported by the Hungarian Scientific Research Fund under the grant number OTKA FK 131793...

Availability of data and materials
The source code of our proposed approach and model is open and available at the following websites: https://github.com/dilshadsallo/DistSysJavaHelpershttps://github.com/dilshadsallo/dissect-cf-examples.

Declarations

Competing interests
The authors declare that they have no competing interests.

Received: 27 August 2022 Accepted: 26 January 2023
Published online: 11 March 2023

References

1. Shafiei H, Khonsari A, Mousavi P (2022) Serverless computing: A survey of opportunities, challenges, and applications. ACM Comput Surv 54(11s):1–32

2. Jounaid S (2020) Opendc serverless: Design, implementation and evaluation of a faas platform simulator. PhD thesis, Vrije Universiteit Amsterdam
3. Hasan M, Siddique MA (2021) A research-oriented mathematical model for cloud simulations. In: 2021 Fifth International Conference on I-SMAC (IoT in Social, Mobile, Analytics and Cloud)(I-SMAC). IEEE, pp 875–878
4. Sallo DH, Kecskemeti G (2022) Towards generating realistic trace for simulating functions-as-a-service. In: European Conference on Parallel Processing. Springer, pp 428–439
5. Calheiros RN, Ranjan R, Beloglazov A, De Rose CA, Buyya R (2011) Cloudsim: a toolkit for modeling and simulation of cloud computing environments and evaluation of resource provisioning algorithms. Softw: Pract Experience 41(1):23–50
6. Kecskemeti G (2015) Dissect-cf: a simulator to foster energy-aware scheduling in infrastructure clouds. Simul Model Pract Theory 58:188–218
7. Buyya R, Murshed M (2002) Gridsim: A toolkit for the modeling and simulation of distributed resource management and scheduling for grid computing. Concurr Comput: Pract Experience 14(13–15):1175–1220
8. Jeon H, Cho C, Shin S, Yoon S (2019) A cloudsim-extension for simulating distributed functions-as-a-service. In: 2019 20th International Conference on parallel and distributed computing, applications and technologies (PDCAT). IEEE, pp 386–391
9. Rausch T, Rashed A, Dustdar S (2021) Optimized container scheduling for data-intensive serverless edge computing. Futur Gener Comput Syst 114:259–271
10. Mahmoudi N, Khazaei H (2021) Simfaas: A performance simulator for serverless computing platforms. arXiv preprint arXiv:2102.08904
11. Shahrad M, Fonseca R, Goiri Í, Chaudhry G, Batum P, Cooke J, Laureano E, Tresness C, Russinovich M, Bianchini R (2020) Serverless in the wild: Characterizing and optimizing the serverless workload at a large cloud provider. In: 2020 {USENIX} Annual Technical Conference ({USENIX}{ATC} 20). arXiv preprint arXiv:2003.03423. pp 205–218
12. Tangherloni A, Spolaor S, Rundo L, Nobile MS, Cazzaniga P, Mauri G, Liò P, Merelli I, Besozzi D (2019) Genhap: a novel computational method based on genetic algorithms for haplotype assembly. BMC Bioinformatics 20(4):1–14
13. Núñez A, Vázquez-Poletti JL, Caminero AC, Castañé GG, Carretero J, Llorente IM (2012) icancloud: A flexible and scalable cloud infrastructure simulator. J Grid Comput 10(1):185–209
14. Chen W, Deelman E (2012) Workflowsim: A toolkit for simulating scientific workflows in distributed environments. In: 2012 IEEE 8th international conference on E-science. IEEE, pp 1–8
15. Tian W, Zhao Y, Xu M, Zhong Y, Sun X (2013) A toolkit for modeling and simulation of real-time virtual machine allocation in a cloud data center. IEEE Trans Autom Sci Eng 12(1):153–161
16. Wickremasinghe B, Calheiros RN, Buyya R (2010) Cloudanalyst: A cloudsim-based visual modeller for analysing cloud computing environments and applications. In: 2010 24th IEEE international conference on advanced information networking and applications. IEEE, pp 446–452
17. Kliazovich D, Bouvry P, Khan SU (2012) Greencloud: a packet-level simulator of energy-aware cloud computing data centers. J Supercomput 62(3):1263–1283
18. Siavashi A, Momtazpour M (2019) Gpucloudsim: an extension of cloudsim for modeling and simulation of gpus in cloud data centers. J Supercomput 75(5):2535–2561
19. Casanova H (2001) Simgrid: A toolkit for the simulation of application scheduling. In: Proceedings First IEEE/ACM International Symposium on Cluster Computing and the Grid. IEEE, pp 430–437
20. Meisner D, Wu J, Wenisch TF (2012) Bighouse: A simulation infrastructure for data center systems. In: 2012 IEEE International Symposium on Performance Analysis of Systems & Software. IEEE, pp 35–45
21. Son J, Dastjerdi AV, Calheiros RN, Ji X, Yoon Y, Buyya R (2015) Cloudsimsdn: Modeling and simulation of software-defined cloud data centers. In: 2015 15th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing. IEEE, pp 475–484
22. Higashino WA, Capretz MA, Bittencourt LF (2015) Cepsim: A simulator for cloud-based complex event processing. In: 2015 IEEE International Congress on Big Data. IEEE, pp 182–190
23. Kohne A, Spohr M, Nagel L, Spinczyk O (2014) Federatedcloudsim: a sla-aware federated cloud simulation framework. In: Proceedings of the 2nd International Workshop on CrossCloud Systems. pp 1–5
24. Calheiros RN, Netto MA, De Rose CA, Buyya R (2013) Emusim: an integrated emulation and simulation environment for modeling, evaluation, and validation of performance of cloud computing applications. Softw: Pract Experience 43(5):595–612
25. Teixeira Sá T, Calheiros RN, Gomes DG (2014) Cloudreports: An extensible simulation tool for energy-aware cloud computing environments. In: cloud computing. Springer, pp 127–142
26. Keller G, Tighe M, Lutfiyya H, Bauer M (2013) Dcsim: A data centre simulation tool. In: 2013 IFIP/IEEE International Symposium on Integrated Network Management (IM 2013). IEEE, pp 1090–1091
27. Cai Z, Li Q, Li X (2017) Elasticsim: A toolkit for simulating workflows with cloud resource runtime auto-scaling and stochastic task execution times. J Grid Comput 15(2):257–272
28. Kathiravelu P, Veiga L (2014) Concurrent and distributed cloudsim simulations. In: 2014 IEEE 22nd International Symposium on Modelling, Analysis & Simulation of Computer and Telecommunication Systems. IEEE, pp 490–493

Publisher's Note

Springer Nature remains neutral with regard to jurisdictional claims in published maps and institutional affiliations.