Edge Workload Trace Gathering and Analysis for Benchmarking

Klervie Toczé
Linköping University, Sweden
klervie.tocze@liu.se

Norbert Schmitt
University of Würzburg, Germany
norbert.schmitt@uni-wuerzburg.de

Ulf Kargén
Linköping University, Sweden
ulf.kargen@liu.se

Atakan Aral
University of Vienna, Austria
atakan.aral@univie.ac.at

Ivona Brandić
Vienna University of Technology, Austria
ivona.brandic@tuwien.ac.at

***Abstract*—The emerging field of edge computing is suffering from a lack of representative data to evaluate rapidly introduced new algorithms or techniques. That is a critical issue as this complex paradigm has numerous different use cases which translate into a highly diverse set of workload types.**

In this work, within the context of the edge computing activity of SPEC RG Cloud, we continue working towards an edge benchmark by defining high-level workload classes as well as collecting and analyzing traces for three real-world edge applications, which, according to the existing literature, are the representatives of those classes. Moreover, we propose a practical and generic methodology for workload definition and gathering. The traces and gathering tool are provided open-source.

In the analysis of the collected workloads, we detect discrepancies between the literature and the traces obtained, thus highlighting the need for a continuing effort into gathering and providing data from real applications, which can be done using the proposed trace gathering methodology. Additionally, we discuss various insights and future directions that rise to the surface through our analysis.

***Index Terms*—Edge/fog workloads, Trace gathering, Edge/fog benchmarking, Open-source.**

I. Introduction

In the past years, the edge computing paradigm has been envisioned as a way to solve some of the issues that are present in the well-established cloud computing paradigm, such as the need for lower latency and for lowering the amount of data sent through the network [1]. Presence of the edge infrastructure is considered as an enabler for an extremely wide range of applications, and there exist numerous works describing what all those applications are or could be [2]–[5]. Concurrently, substantial research effort is made to provide solutions for the challenges brought by the edge paradigm, such as resource management over large sets of distributed and heterogeneous devices, workload scheduling/orchestration, and security [6].

Therefore, it is crucial for the research pursued to evaluate the algorithms and techniques created on relevant workloads. Otherwise, there is a risk that the community meticulously creates theoretical solutions that might underperform in practice. Getting access to such workloads is not easy, either because they do not exist yet, the access to available applications is restricted due to business interests, or privacy obligations and the hardness of data anonymization impede the release of collected data [7]. Consequently, most of the edge benchmarking approaches in the literature [8] rely on traces obtained from simulators, which conflicts with the purpose of benchmarking. Above-mentioned accessibility problem motivates our work, which aims to create realistic and relevant workloads for the edge computing community to test and evaluate the new algorithms and techniques. The long-term goal is to propose an edge benchmark for the same purpose.

The need for more edge benchmarks has been already highlighted and there are ongoing efforts [8]. For example, for benchmarking edge computing platforms [9], application performance over different edge platforms [10], or augmented reality and wearable cognitive assistance applications [11]. Our work differs from these in that we focus on the applications rather than the hardware and that we follow a systematic methodology to classify edge computing workloads and to identify the most representative ones.

In this paper, we build on our preliminary work [12], which presented a set of workload characteristics and a preliminary characterization of three use cases. In this work, we deepen the analysis of the edge computing application landscape by considering 24 applications. This analysis enables the definition of a small set of representative workload classes to work with, instead of blindly trying to cover the wide range of applications described in the literature. Then, we look at what applications are currently available and relevant for the defined workload classes and collect data from them. We make the data traces publicly available, and present an analysis in this paper. Finally, we discuss insights gained, how the collected traces can be used as well as other issues. This work thus covers the first two steps highlighted in Figure 1 towards an edge benchmark for algorithms and techniques. The details of these two steps as well as our contributions are provided in the rest of this paper and illustrated in Figure 2.

Accordingly, the research questions considered in this work are: 1) How can current and envisioned edge computing

Klervie Toczé is supported by the Swedish national graduate school in computer science (CUGS). Atakan Aral and Ivona Brandić are supported by the CHIST-ERA grant CHIST-ERA-19-CES-005, and by the Austrian Science Fund (FWF): projects Y904-N31 (RUCON) and I5201-N (SWAIN).

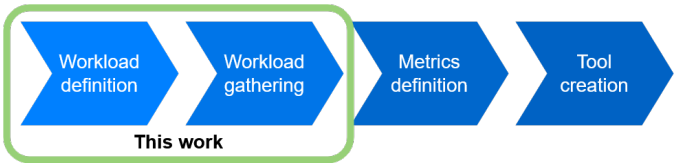

Fig. 1. Steps for creating an edge benchmark for algorithms and techniques.

applications be classified according to workload characteristics relevant for evaluating algorithms and techniques? 2) How do currently available workloads for those classes look like? 3) How can knowledge about current workloads be used for pertinent evaluation of edge algorithms and techniques?

In order to provide answers to those questions, the following are the contributions of this paper.

- An extended use case characterization study leading to the definition of high-level workload classes.
- A generic methodology for trace gathering including an instruction counting tool.
- Three open-source workload traces together with an analysis and discussion of these.
- A discussion on insights, challenges, and directions.

We believe that these contributions would aid researchers and practitioners alike in better understanding current edge workloads, as well as enabling the comparative evaluation of edge algorithms and techniques for these workloads. Openness and flexibility of the provided tools and methods would also foster future classification attempts for edge workloads.

This paper is structured as follows: we present related work in Section II. Then, in Section III, we characterize an extended set of edge use cases and define four workload classes. We describe the process of gathering traces from available applications in Section IV, and analyze the results obtained in Section V. Finally, we provide a discussion of our findings in Section VI, and conclude the paper in Section VII.

II. Related Work

While numerous surveys on the edge paradigm include a part about edge applications (e.g. [5], [13], [14]), some have a deeper focus on these. For example, Liu et al. [15] propose an edge application classification according to three aspects. The first one is the application *properties*, e.g. latency-sensitivity or local context-awareness. The second is the *type* of application, as edge computing can be used for traditional applications (e.g. client-server) but also for emerging ones (e.g. IoT). Finally, they consider the *interaction* between the edge and the user, which can be for example computation offloading or collaborative computing.

Ahmed et al. [16] studied 30 representative fog computing applications described in the literature with regards to 17 different aspects, including why they use fog, the deployment model used, type of access network, type of fog nodes, latency-sensitivity, security, and workload characteristic (stable vs dynamic). However, the above works only present existing or envisioned applications, without studying them further. In this work, we first analyze a subset of edge applications to then move forward into looking at existing applications in order to collect data that will enable further experimental studies.

Although still in its infancy, some efforts are ongoing in the field of edge and fog benchmarking. Varghese et al. [8] presented a systematic classification of edge performance benchmarking in the three dimensions of (i) system under test (infrastructure, e.g., CPU, memory, etc. and software platform, e.g., orchestrators, schedulers, etc.), (ii) techniques analyzed (optimization and deployment options), and (iii) benchmark runtime (software and data characteristics). Surveyed benchmarks date from as early as 2014. Silva et al. [17] proposed a methodology for creating benchmarks for edge frameworks for stream processing (e.g., Apache Edgent, Amazon Greengrass, or Azure stream analytics). They used data from the New York City taxi data set and CCTV footage from the university of California San Diego as application workload. Those data sets are only the input data and do not provide information about the actual edge workload, as in this paper. Focusing on deployment platforms and modes, McChesney et al. [10] proposed the DeFog benchmarking suite. It includes six applications of different types that can be studied according to cloud-only, edge-only or cloud-edge deployment. The different applications are said to exhibit different characteristics but no further analysis is provided, contrary to this work.

There are also efforts on generating traces for various properties of edge infrastructures. Aral et al. [18] focused on reliability and proposed the fusion of failure traces collected from three relevant systems that represent different deployment strategies proposed in the edge computing literature. More recently, Kolosov et al. [19] discussed thoroughly the current lack of comprehensive data sets for edge computing, which is impeding benchmarking efforts. They provide a list of attributes needed in the long term in those data sets and propose the concept of workload composition for creating comprehensive data sets based on the partial ones that are currently available. Our work contributes to providing compute-related workloads, which according to their studies are one of the least available at the moment.

III. Workload classification

The first step towards an edge benchmark (see Figure 1) consists in defining what type(s) of workload should be included. Indeed, there is a large number of different edge applications, but collecting traces for all the possible applications is practically infeasible, especially since most of them are only *envisioned* at the moment, with no real implementation. By gathering applications with similar characteristics into the same workload class, the number of applications required for trace gathering becomes substantially smaller.

A. Methodology and scope

To create the edge workload classes, the following methodology is used (illustrated in Figure 2, upper part). First, we define the scope of the classes, i.e., for what purposes the workloads will be used. This is critical as edge computing

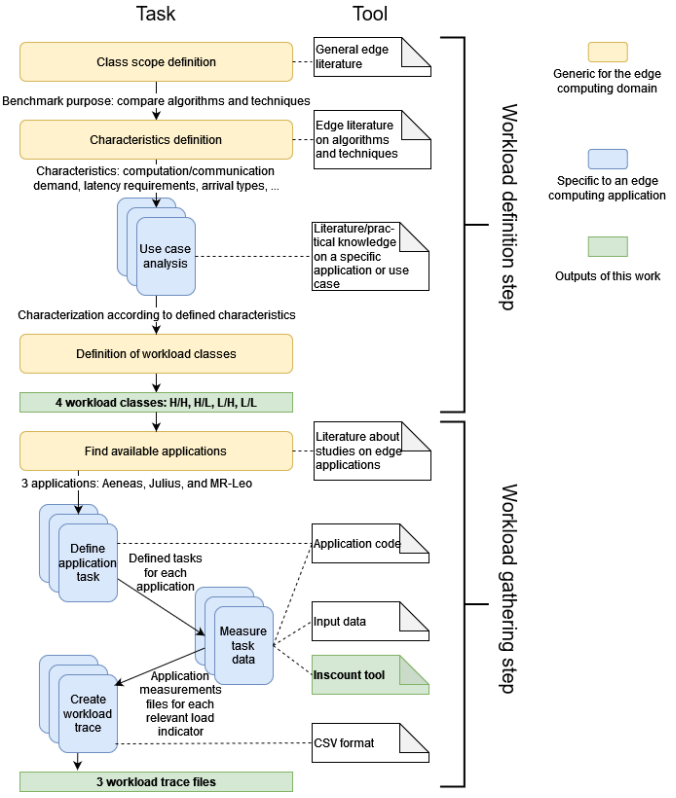

Fig. 2. Workload definition and gathering methodology.

work can be performed at different levels and with different angles [19]. Secondly, we define characteristics that are relevant to include when gathering workload data, based on the scope. Finally, we look at use case descriptions in the literature to analyze a range of use cases with regards to the different characteristics. The aim was to identify the most relevant characteristics for creating edge workload classes.

In this work, we focus on workloads that can be used for evaluating edge algorithms or techniques, such as orchestration or placement algorithms. The aim is to obtain knowledge about the work that the edge system will have to execute in order to provide relevant techniques for performing this execution in an as efficient way as possible. The scope of this work thus falls into the *compute-related* workload category as defined by Kolosov et al. [19], and *orchestrators* and *schedulers* categories as defined by Varghese et al. [8].

B. Extended use case characterization

We characterize an extended set of use cases from the edge computing literature using the following methodology (from our previous work [12]). Six characteristics are included: three regarding resource demand (computation, communication, and storage), latency requirement (previously called deadline[1]), arrival type, and interarrival time. For each characteristic, two nominal classes are defined to categorize the use cases (e.g. high/low, short/long). The complete characterization of the 24 studied use cases is made available as a public archive [20]. The use cases include among others mixed reality, event-driven applications, medical applications, and smart city use cases.

C. Workload classes

When studying which characteristics are put forward by the literature describing the use cases in the above characterization step, it appears that it is either the computation demand, the communication demand or the latency requirements[2]. Even if it is an important aspect in edge computing, not all edge use cases have defined latency requirements, but all of them require computation and communication resources. Therefore, the workload classes are defined according to the possible combinations of nominal classes for those two characteristics. Focusing on only two characteristics also allows for a limited set of classes, which is of interest since the number of available edge applications for studies is currently limited. Four workload classes, namely **H/H**, **H/L**, **L/H**, and **L/L** are thus defined as shown in Table I.

TABLE I
DEFINITION OF EDGE WORKLOAD CLASSES.

		Computation Demand	
		Low	High
Communication Demand	Low	**L/L**	**H/L**
	High	**L/H**	**H/H**

IV. TRACE GATHERING

The second step towards an edge benchmark as shown in Figure 1 is to gather workload data, e.g. workload traces.

A. Methodology

The trace gathering methodology is illustrated in the lower part of Figure 2. To gather application data, it is first needed to find applications that are available open-source and fit into one of the defined workload classes according to their description in the corresponding publications.

Once a selection of applications is made, these are tested and analyzed in order to define what a *task* corresponds to in the context of this particular application. A task should consists of a) input data, b) computation to be performed on the input data, and c) output data. It should be possible to isolate in the edge application code where a task starts executing and when

[1]In this work, we renamed the *deadline* characteristic to *latency requirement* in order to better reflect the fact that even if some use cases do not have a formally defined deadline, there is an expectation for how long a task can take to complete. The two categories for this characteristic are Close (the expected latency ranges from tens of milliseconds to tens of seconds), and Far (the expected latency is higher than tens of seconds).

[2]This does not mean that the application cannot have requirements on multiple characteristics, but that one of them is emphasized.

it finishes to collect data about its computational resource demand. In this study, we consider that the application code is available on the edge server (e.g. inside a container).

Based on the characteristics of Section III, the traces should contain data about resource demand (especially computation and communication), latency requirements, and task arrivals.

This trace gathering process is generic and can be used with any application as long as it is possible to define a task and instrument the code at the start and end of it[3].

B. Selected applications

The following applications are selected for the different workload classes, based on their description in the literature and their open-source availability[4]:

- **H/H**: Mixed reality (MR-Leo)
- **L/H**: Forced alignment (Aeneas)
- **L/L**: Speech recognition (Julius)

1) Forced alignment: Forced alignment focuses on synchronizing audio and text. For an input text and the corresponding audio, it aims to find the time interval in the audio that corresponds to a text fragment. The output is a synchronization map between the text and audio. This application type is already used in edge/fog benchmarking [10], as edge computing increases its responsiveness. We use Aeneas[5] as an implementation of the forced alignment use case. It is an open-source Python/C library that works on MacOS, Windows, and Linux and supports a large variety of input/output file types.

2) Speech recognition: Speech recognition uses different models and methods to translate spoken language into writing or executable commands. Speech recognition tasks are gaining popularity and can be executed on hardware with relatively low computational power, depending on the model and methods used. Latency is the central requirement because speech recognition should be done within a reasonable timeframe. We use the Julius library[6] as the implementation for the speech recognition use case. It claims a low memory consumption for its implementation and low computation demand [21], [22], making it suitable for a broad range of systems under test.

3) Mixed reality: Mixed reality (MR) integrates virtual components into a scene from the reality (in that case it is also called augmented reality) or vice-versa (augmented virtuality). An MR application can be decomposed into the following steps: 1) Capture of the real scene, 2) Analysis of the real scene to construct a 3D map or detect 3D objects inside the scene, 3) Addition of the virtual elements or integration into the virtual scene, and 4) Display on the end user device. In an edge context, steps 2 and 3 are fully or partially offloaded to an edge device. We use MR-Leo [23] as the implementation for the mixed reality use case. It is an open-source prototype that fully offloads steps 2 and 3 described above. The end user part of the application is an Android application and the edge server part is a C program running on Linux.

C. Traces gathered

1) Gathering process overview: The input and output data for each application example is isolated and its size measured in order to get the communication demand. For the computation demand, it is measured in number of instructions using a tool that we developed, called Inscount. Finally, information about the task arrival time (relative to the start of the trace) and the task deadline are also collected when this is relevant.

The traces are uploaded to Zenodo and made available open-source [24] so that the community can use them for performing further edge studies. As access to an edge infrastructure deployment is complicated at the moment, the traces are collected on research edge setups that are representative of the type of devices that can be used as edge devices. More details about these are provided on Zenodo. As the applications, input data, and gathering tool are available open-source, it will be possible to extend this study to future edge infrastructures.

2) Trace format: All the traces gathered in this work consist of a comma-separated values (CSV) file for each application. The available fields are: task timestamp, task id, application type, computation demand (in number of instructions), communication demand for the uplink (in bytes), communication demand for the downlink (in bytes), and task deadline. The task timestamp and task deadline fields are optional and are only filled when relevant to the particular application.

3) The Inscount tool: In order to characterize the computational demand using a metric that is not dependent on the actual CPU used during the gathering process (such as CPU time), we opted for measuring instruction counts. While there exist several profiling tools that can count the total number of executed instructions within, for example, one function, our application requires measuring instruction counts between precisely defined points of program execution. Therefore, we implemented our own instruction counting tool called *Inscount*, based on the Pin [25] dynamic binary instrumentation framework. The program to be traced must be recompiled to insert calls to two dummy functions, which marks the start and end points of instruction counting. The Inscount tool will monitor for calls to these functions, and record the precise instruction count for the corresponding interval of execution. Note that, since instrumentation is performed on the binary level, we can effortlessly count instructions in, e.g., third-party libraries, for which source code may not be available. The tool also supports tracing multi-threaded code by serializing execution on the basic block level. In the interest of open science, we make the Inscount tool available as open source[7].

4) Aeneas: For collecting the trace for the Aeneas application, 20 out of the 36 example scenarios included in the application are randomly selected. These consist in different variants of text input file and output format for the same audio file (a 53.240 seconds recording of the *Sonnet I* by William

[3]In this paper, we provide Inscount for instrumenting programs running on Intel processors, other processor architectures will require other tools.

[4]At the time of the study, no open-source edge application for the H/L class could be found.

[5]https://www.readbeyond.it/aeneas/

[6]https://github.com/julius-speech/julius

[7]https://gitlab.liu.se/ulfka17/inscount

Shakespeare). Each selected example is run 30 times. A task for Aeneas consists in taking as input an audio file and a text file, performing the forced aligment algorithm and returning an output file containing the alignment. A task is sent whenever needed and will take more or less time to complete mostly based on the length of the text/audio to align. Therefore, the trace does not include task timestamp or task deadline, as those are not relevant for Aeneas.

5) Julius: The Julius trace is collected using 20 audio files from the spoken Wikipedia corpora[8] as input, containing 1314 articles in 2862 audio files. Among these files, 20 were selected at random without duplicates. Each input was run 30 times on four servers. The order of execution was shuffled after each iteration. A task for Julius consists of an audio file as input, on which the speech recognition is performed and outputs the recognized sentences to the console. Similar to Aeneas, a task is sent whenever needed, hence the trace does not include task timestamp and deadline. For the communication downlink, the size (in bytes) of the original text from the corresponding article coming with the spoken Wikipedia corpora is used. This is due to Julius outputting status and progress reports to the console intermixed with the detected sentences and words, hence making this output different from the one the user would expect to receive. Choosing the original text is the optimal output Julius can produce. Erroneously detected words would not change the output size drastically and the original text is, therefore, a good approximation.

6) MR-Leo: The MR-Leo trace is gathered using three different input videos[9]. The length of each video is one minute but they are looped to allow for trace gathering for different video lengths. The three videos are run five times each. The actual use case is live video streaming, however it is impractical to gather data for it due to the slow down incurred by the Inscount tool. A task for MR-Leo consist in a video frame that has to be analyzed by the MR framework and the output is an MR-enhanced video frame. Video frames arrive at a defined rate and should be analyzed before the next one comes in order to avoid delay that will degrade the user quality of service [26]. Task timestamp and task deadline are thus included in the trace.

V. Trace analysis

We now analyze the traces collected in Section IV and present insights about the selected application workloads.

A. Task definition

With regards to how a task is defined for each application, we identify two groups according the collected traces. The first group includes the Aeneas and Julius traces and corresponds to the applications where there are no set requirements yet for when a task is sent and how long it should take to perform. This is because there is no commercial product using these specific applications currently, to the best of our knowledge. The second group consists of the MR-Leo trace, where there are expectations on task arrivals and task deadlines in order to maintain a high QoS.

[8]https://nats.gitlab.io/swc/

[9]Available at https://gitlab.liu.se/ida-rtslab/public-code/2019_mrleo_video and https://gitlab.liu.se/ida-rtslab/public-code/2019_MRLEO_charac

TABLE II
Metric Summary (Min-Max) for Different Application Traces

	# instructions per input byte	Input size in bytes	Output size in bytes
Aeneas	2113–208139	426735–427467	324–32130
Julius	52676–62911	660132–10325884	1482–76396
MR-Leo	3605–5629	41000	41000

B. Computation

Box plots of the number of instructions measured per task are generated. We note that although Aeneas and Julius were selected for workload classes having a lower computational demand compared to MR-Leo, the situation is the opposite after analyzing the traces. As the three traces were gathered using different type of input files having varying sizes, it is however unfair to compare the instructions numbers as such. For instance, taking the full video as a single task for MR-Leo would yield a significantly higher number of instructions per task. Therefore, we choose to instead present the number of instructions per input byte in order to visualize how intensive the computation is for each byte of input data received at the edge. These results are presented in Figures 3, 4, and 5, where different examples executed are shown on the x-axis. Table II shows a summary of the results, including the minimum and maximum values over all examples.

This metric shows that how compute-intensive an application is actually depends on the scenario it is used for. This is especially visible in the Aeneas case where the number of instruction per input byte can vary by almost a factor 100 depending on the example considered. Julius and MR-Leo, on the other hand, show a more stable behavior with a lower variation between the different examples selected. Another observation is that the difference in number of instructions measured in different runs using the same data is very low for Aeneas and Julius, while it is higher for MR-Leo. This is due to the internal mechanisms of the applications and whether they handle the same input in a deterministic way.

C. Communication

With regards to communication, the traces record the size of the required input, as well as of the output for the three applications considered. Figures 6,7, and 8 present our results and Table II summarizes the ranges for the input and output of the different applications. For Aeneas and Julius, the input and output sizes depend on the example considered (e.g. different files or file formats are used) whereas MR-Leo has a constant input and output size as all video frames have the same size in the examples chosen.

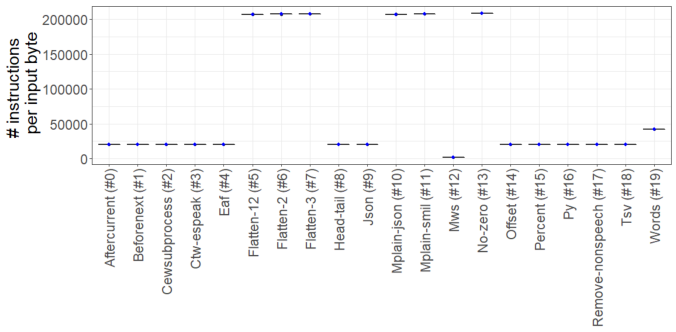

Fig. 3. Number of instructions per input byte measured for Aeneas tasks.

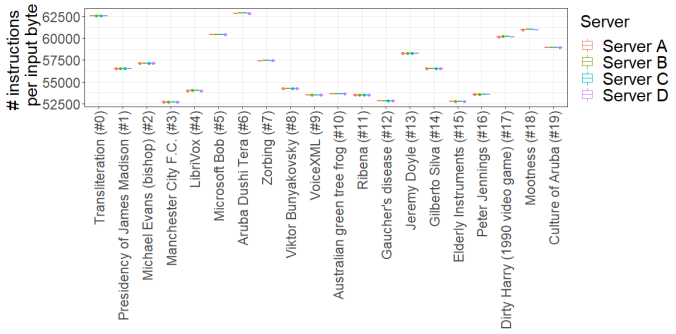

Fig. 4. Number of instructions per input byte measured for Julius tasks on the four different servers.

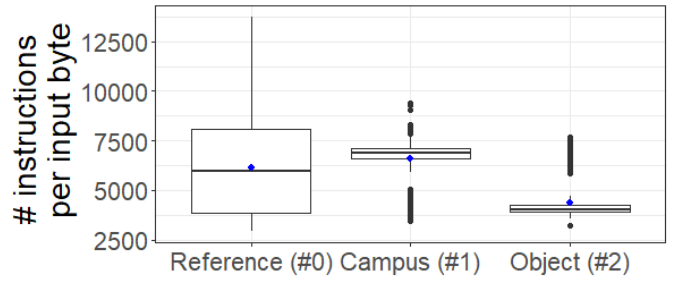

Fig. 5. Number of instructions per input byte measured for MR-Leo tasks.

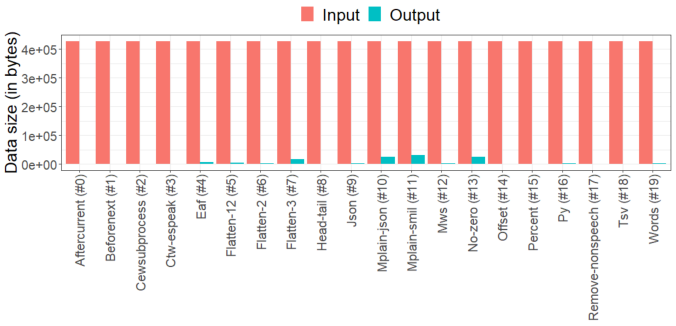

Fig. 6. Size of input and output data for Aeneas tasks.

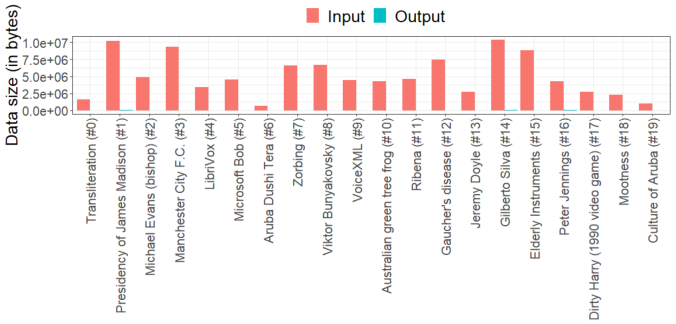

Fig. 7. Size of input and output data for Julius tasks.

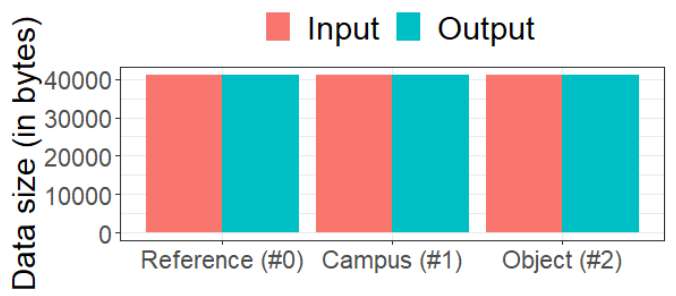

Fig. 8. Size of input and output data for MR-Leo tasks.

An important observation is that the range of input or output sizes is broad (up to almost 100-fold for the Aeneas output), thus highlighting the important role the actual example selected will have on how the load of these applications will look like. Depending on the input chosen, the same application could appear as being very communication-intensive or quite the opposite. MR-Leo has no range as all examples tested had the same resolution, but similar behaviour is expected if testing with other video resolutions. Another observation is that applications such as Aeneas or Julius are a lot more communication-intensive in the uplink than in the downlink (e.g. 210 times more for the Julius task #14), as transferring audio files is heavier than text files.

VI. Discussion

Along the course of this work, various insights were gained related to edge workloads and how to use them. In this section, we discuss these as well as current challenges and future directions for an edge computing benchmark.

A. Edge workloads in theory and in practice

The first important insight from our study is that there is currently a gap between how applications are described in the literature and how the (rare) implementations available behave. This is visible when looking at how computationally-intensive the different applications selected should be and how actually intensive they were, which did not completely match. The same behavior is observable for the communication intensity,

where the same application could be classified as intensive or not depending on the test example chosen. Obviously, this is highly dependent on the applications chosen for trace gathering and it can very well be that those specific applications behave in a specific way that might not be representative for how this application type will behave generally in the future. However, there is currently a scarcity of edge computing applications that are available open-source; thus, our findings apply to the current state of the technology.

Therefore, it is of high importance to encourage the development and open-source release of different types of edge applications in order to increase the number of traces available. This would provide an accurate picture of the edge workloads to researchers or practitioners working on algorithms or techniques to provide an efficient handling of those workloads. An inaccurate workload picture based on only description of how use cases are expected to behave could potentially lead to research effort not focusing on the actual bottlenecks, which is a waste of time and resources for the community and could even impede the deployment of the edge paradigm.

Our generic trace gathering methodology is designed in a way that it can be easily reused when more applications are available. This will enable the community to provide traces that correspond to the high-level workload classes and can be included in a future edge benchmark.

B. Towards edge benchmarking

This work is conducted within the scope of the edge computing activity of SPEC Research Group Cloud. It is a part of an ongoing effort to provide an edge benchmark that can be used to evaluate edge algorithms and techniques on a set of workloads exhibiting different characteristics of interests, i.e. from the different workload classes proposed. The approach that is taken was to select one application per class, as they, according to the available literature, should exhibit those different characteristics. However, the trace gathering and analysis performed shows that this approach does not produce the expected results, as the same application can produce workloads corresponding to different classes based on the input considered. Therefore, instead of selecting one application per workload class, it is advisable to select both applications and appropriate sets of input for each workload class when creating a future benchmark. Doing so will also allow for different type of applications to be available for each workload class, which increases the benchmark relevance.

C. Leveraging workload traces

The traces gathered in this work focus on providing compute-related data about tasks that are sent to the edge for execution. They can be immediately used for benchmarking algorithms that are proposed for resource management in edge computing paradigm. These may include, schedulers, replication engines, resource provisioning, resource allocation, etc. The traces can be, for example, input to the workload generator of a simulator to evaluate a task placement algorithm.

Nevertheless, additional data sets that cover attributes that relate to storage, communication, user location and mobility, etc. are required for getting the full picture of an edge workload. However, it is currently not possible to gather all those attributes as the edge systems that would enable such gathering are not yet available. Kolosov et al. [19] propose the concept of workload composition in order to construct relevant data sets in the meantime, where actual full-fledge data sets are not available. The work presented here fits into this idea and enlarges the body of the compute-related data that is one of the those with fewest available traces. As a practical example of workload composition, Aral et al. [27] combined real-world mobility traces from electric vehicles with the simulated edge computing performance traces in their study on running data analytics on mobile devices and vehicles. The availability of more traces will promote such joint uses of data sets for a more complete picture of the edge computing ecosystem.

Another possibility to leverage the traces provided in this paper is by applying the transfer learning concept. Data sets that are available to the research community are more common for the distributed systems that are architecturally similar to edge computing but are more established, such as cloud computing or peer-to-peer systems. Hence, an interesting solution to data set scarcity is to adapt data sets from closely associated domains and emulate edge computing data traces. There exists a comprehensive body of knowledge for domain adaptation in the context of transfer learning [28]. Transfer learning is typically used to improve a learner in the target domain (e.g. edge computing) by transferring information from a related source domain (e.g. cloud computing). Most transfer learning techniques generate an adapted training data set from the source domain data that is suitable for or representative of the target domain. This approach has shown to be highly effective in many application areas summarized in literature reviews [28], [29]. One idea for further studies is to use the traces collected in this work to validate the transfer learning concept applied to edge computing.

D. Open Issue: Upscaling the benchmarks

Above described techniques are intended to leverage workload traces vertically, that is generating additional traces for a single instance of an edge computing application. However, a particular aspect of these applications is that they are usually replicated in many geographically distributed edge nodes to exploit proximity to data sources and consumers. Consequently, each instance of the application behaves differently based on the local conditions such as available compute resources or input data characteristics. These non-trivial discrepancies should also be reflected in benchmarks.

Consider, for example, the recent application of real-time video analytics to be deployed at smart traffic lights in the city of Vienna [30]. This work, which aims at improving traffic safety by warning drivers about pedestrians and cyclists in their blind spots, has been already tested in a junction, where substantial workload traces were collected. Large-scale performance of the application at 1343 traffic lights throughout

Vienna is yet to be evaluated. The questions is then how to create semi-artificial data sets from a single traffic light for others considering their local characteristics. These characteristics might include pedestrian and vehicle traffic density, which affects the data frequency, or 5G coverage, which affects the communication latency.

VII. Conclusion

In this paper, we focus on the first two steps required to create an edge benchmark. First, we perform an extended characterization of a variety of edge use cases and propose workload classes in order to help categorizing the vast number of edge use cases. Then, we select three applications that, according to the literature, correspond to three different workload classes and gather data related to their workload, which we make available open-source. An analysis of the data collected clearly demonstrates that referring only the literature is not suitable for classifying the rare edge applications available for study and that one application can exhibit characteristics from different workload classes based on the inputs considered. Therefore, more application development, trace gathering, and analysis (e.g., statistical) is needed to provide data sets corresponding to the high-level workload classes.

Future extension possibilities include the gathering of traces for more applications with the goal of creating an edge benchmark providing relevant data input for researchers and practitioners who need to evaluate their edge algorithms or techniques. This will require to develop Inscount-like tools for other processor types. Other directions include applying workload composition and transfer learning to provide data sets covering more attributes or application scenarios.

References

[1] M. Satyanarayanan, "The emergence of edge computing," *Computer*, vol. 50, no. 1, pp. 30–39, Jan 2017.

[2] C. Perera, Y. Qin, J. C. Estrella, S. Reiff-Marganiec, and A. V. Vasilakos, "Fog computing for sustainable smart cities: A survey," *ACM Comput. Surv.*, vol. 50, no. 3, pp. 32:1–32:43, Jun. 2017.

[3] M. T. Beck, M. Werner, S. Feld, and T. Schimper, "Mobile edge computing: A taxonomy," in *Proc. of the Sixth International Conference on Advances in Future Internet*. Citeseer, 2014, pp. 48–55.

[4] R. K. Naha, S. Garg, D. Georgakopoulos, P. P. Jayaraman, L. Gao, Y. Xiang, and R. Ranjan, "Fog computing: Survey of trends, architectures, requirements, and research directions," *IEEE Access*, vol. 6, pp. 47 980–48 009, August 2018.

[5] N. Abbas, Y. Zhang, A. Taherkordi, and T. Skeie, "Mobile edge computing: A survey," *IEEE Internet of Things Journal*, vol. 5, no. 1, pp. 450–465, February 2018.

[6] A. J. Ferrer, J. M. Marquès, and J. Jorba, "Towards the decentralised cloud: Survey on approaches and challenges for mobile, ad hoc, and edge computing," *ACM Comput. Surv.*, vol. 51, no. 6, Jan. 2019.

[7] C. Reiss, J. Wilkes, and J. L. Hellerstein, "Obfuscatory obscanturism: Making workload traces of commercially-sensitive systems safe to release," in *IEEE Network Operations and Management Symposium*, 2012, pp. 1279–1286.

[8] B. Varghese, N. Wang, D. Bermbach, C.-H. Hong, E. D. Lara, W. Shi, and C. Stewart, "A survey on edge performance benchmarking," *ACM Computing Surveys (CSUR)*, vol. 54, no. 3, pp. 1–33, 2021.

[9] A. Das, S. Patterson, and M. Wittie, "Edgebench: Benchmarking edge computing platforms," in *IEEE/ACM International Conference on Utility and Cloud Computing (UCC) Companion*, Dec 2018, pp. 175–180.

[10] J. McChesney, N. Wang, A. Tanwer, E. de Lara, and B. Varghese, "Defog: Fog computing benchmarks," in *ACM/IEEE Symposium on Edge Computing*, ser. SEC '19. ACM, 2019, p. 47–58.

[11] M. O. J. Olguín Muñoz, J. Wang, M. Satyanarayanan, and J. Gross, "Edgedroid: An experimental approach to benchmarking human-in-the-loop applications," in *International Workshop on Mobile Computing Systems and Applications (HotMobile '19)*. ACM, 2019, pp. 93–98.

[12] K. Toczé, N. Schmitt, I. Brandic, A. Aral, and S. Nadjm-Tehrani, "Towards edge benchmarking: A methodology for characterizing edge workloads," in *IEEE 4th International Workshops on Foundations and Applications of Self* Systems (FAS*W)*, 2019, pp. 70–71.

[13] A. C. Baktir, A. Ozgovde, and C. Ersoy, "How can edge computing benefit from software-defined networking: A survey, use cases, and future directions," *IEEE Communications Surveys Tutorials*, vol. 19, no. 4, pp. 2359–2391, June 2017.

[14] M. Heck, J. Edinger, D. Schaefer, and C. Becker, "IoT applications in fog and edge computing: Where are we and where are we going?" in *2018 27th International Conference on Computer Communication and Networks (ICCCN)*, 2018, pp. 1–6.

[15] H. Liu, F. Eldarrat, H. Alqahtani, A. Reznik, X. de Foy, and Y. Zhang, "Mobile edge cloud system: Architectures, challenges, and approaches," *IEEE Systems Journal*, pp. 2495–2508, September 2017.

[16] A. Ahmed, H. Arkian, D. Battulga, A. J. Fahs, M. Farhadi, D. Giouroukis, A. Gougeon, F. O. Gutierrez, G. Pierre, P. R. S. Jr, M. A. Tamiru, and L. Wu, "Fog computing applications: Taxonomy and requirements," 2019.

[17] P. Silva, A. Costan, and G. Antoniu, "Towards a methodology for benchmarking edge processing frameworks," in *2019 IEEE International Parallel and Distributed Processing Symposium Workshops (IPDPSW)*, 2019, pp. 904–907.

[18] A. Aral and I. Brandić, "Learning spatiotemporal failure dependencies for resilient edge computing services," *IEEE Transactions on Parallel and Distributed Systems*, vol. 32, no. 7, pp. 1578–1590, 2020.

[19] O. Kolosov, G. Yadgar, S. Maheshwari, and E. Soljanin, "Benchmarking in the dark: On the absence of comprehensive edge datasets," in *3rd USENIX Workshop on Hot Topics in Edge Computing (HotEdge 20)*. USENIX Association, Jun. 2020.

[20] K. Toczé, N. Schmitt, U. Kargén, A. Aral, and I. Brandić, "Extended edge use case characterization," January 2022. [Online]. Available: https://doi.org/10.5281/zenodo.5782871

[21] A. Lee, T. Kawahara, and K. Shikano, "Julius—an open source real-time large vocabulary recognition engine," in *European Conference on Speech Communication and Technology (Eurospeech)*, 2001, pp. 1691–1694.

[22] A. Lee and T. Kawahara, "Recent development of open-source speech recognition engine julius," in *Proceedings: APSIPA ASC 2009: Asia-Pacific Signal and Information Processing Association, 2009 Annual Summit and Conference*, 2009, pp. 131–137.

[23] K. Toczé, J. Lindqvist, and S. Nadjm-Tehrani, "Performance study of mixed reality for edge computing," in *IEEE/ACM International Conference on Utility and Cloud Computing*, 2019, p. 285–294.

[24] K. Toczé, N. Schmitt, U. Kargén, A. Aral, and I. Brandić, "Edge workload traces from Aeneas, Julius, and MR-Leo," August 2020. [Online]. Available: https://doi.org/10.5281/zenodo.3974220

[25] C.-K. Luk, R. Cohn, R. Muth, H. Patil, A. Klauser, G. Lowney, S. Wallace, V. J. Reddi, and K. Hazelwood, "Pin: Building customized program analysis tools with dynamic instrumentation," *SIGPLAN Not.*, vol. 40, no. 6, p. 190–200, Jun. 2005.

[26] K. Toczé, J. Lindqvist, and S. Nadjm-Tehrani, "Characterization and modeling of an edge computing mixed reality workload," *Journal of Cloud Computing*, vol. 9, no. 46, 2020.

[27] A. Aral, M. Erol-Kantarci, and I. Brandić, "Staleness control for edge data analytics," *Proceedings of the ACM on Measurement and Analysis of Computing Systems*, vol. 4, no. 2, pp. 1–24, 2020.

[28] S. J. Pan and Q. Yang, "A survey on transfer learning," *IEEE Transactions on knowledge and data engineering*, vol. 22, no. 10, pp. 1345–1359, 2009.

[29] K. Weiss, T. M. Khoshgoftaar, and D. Wang, "A survey of transfer learning," *Journal of Big Data*, vol. 3, no. 1, p. 9, 2016.

[30] I. Lujic, V. D. Maio, K. Pollhammer, I. Bodrozic, J. Lasic, and I. Brandic, "Increasing traffic safety with real-time edge analytics and 5G," in *International Workshop on Edge Systems, Analytics and Networking*, 2021, pp. 19–24.