Large-scale Graph Processing and Simulation with Serverless Workflows in Federated FaaS

Sashko Ristov
sashko.ristov@uibk.ac.at
University of Innsbruck
Innsbruck, Austria

Reza Farahani
reza.farahani@aau.at
Alpen-Adria-Universität Klagenfurt
Klagenfurt, Austria

Radu Prodan
radu.prodan@aau.at
Alpen-Adria-Universität Klagenfurt
Klagenfurt, Austria

ABSTRACT

Serverless computing offers an affordable and easy way to code lightweight functions that can be invoked based on some events to perform simple tasks. For more complicated processing, multiple serverless functions can be orchestrated as a directed acyclic graph to form a serverless workflow, so-called *function choreography* (FC). Although most famous cloud providers offer FC management systems such as *AWS Step Functions*, and there are also several open-source FC management systems (*e.g.*, *Apache OpenWhisk*), their primary focus is on describing the control flow and data flow between serverless functions in the FC. Moreover, the existing FC management systems rarely consider the processed data, which is commonly represented in a graph format. In this paper, we review the capabilities of the existing FC management systems in supporting graph processing applications. We also raise two key research questions related to large-scale graph processing using serverless computing in federated Function-as-a-Service (FaaS). As part of the *Graph-Massivizer* project, funded by the Horizon Europe research and innovation program, we will research and develop (prototype) solutions that will address these challenges.

CCS CONCEPTS

• **Theory of computation → Graph algorithms analysis**; • **Computer systems organization → Cloud computing**.

KEYWORDS

Computing Continuum; Serverless Computing; Graph Processing; Massive Graph; Workflows.

ACM Reference Format:
Sashko Ristov, Reza Farahani, and Radu Prodan. 2023. Large-scale Graph Processing and Simulation with Serverless Workflows in Federated FaaS. In *Companion of the 2023 ACM/SPEC International Conference on Performance Engineering (ICPE '23 Companion), April 15–19, 2023, Coimbra, Portugal.* ACM, New York, NY, USA, 5 pages. https://doi.org/10.1145/3578245.3585333

1 INTRODUCTION

Serverless computing is becoming the de-facto technology to transform the future of computing due to offering the real "pay-as-you-go" charging model at the finest grain. More and more cloud providers *serverlessize* their cloud services; Among others, many cloud providers offer serverless containers (*e.g.*, AWS Fargate or Google Cloud Run), serverless databases (*e.g.*, AWS DynamoDB), and even serverless graph processing (*e.g.*, AWS Neptune).

The most common serverless computing cloud service is *Function-as-a-Service* (FaaS), which offers developers to code their applications in the form of *serverless functions*. In FaaS-enabled systems, the entire infrastructure and platform management, including scaling up and down decisions, is performed by the cloud providers, while developers only upload their codes and invoke the functions. Cloud providers usually do not charge for functions' deployment, charging solely for the resources utilized during the function's runtime. For instance, AWS charges their AWS Lambda users to the closest millisecond. At the same time, AWS Lambda serverless functions can be configured within a range of 128 MB up to 10 GB RAM memory, with a fine-grained memory precision of 1 MB.

Many scientists have used FaaS in recent years to propose scientific computing solutions [16, 30, 41]. Complex applications are built in the form of *serverless workflows* or also called *function choreographies* (FCs) [36] by orchestrating serverless functions. Various FCs exist in the literature for batch [36], event-based [2], or real-time processing [19]. The top five cloud providers offer *FC management systems*, as well, such as AWS Step Functions, Google Workflows, or IBM Composer. Apart from the FC management systems of public cloud providers, researchers introduced many open-source FC management systems, such as xAFCL [37], LithOps [40], and PyWren [21]. All these systems offer proprietary JSON, YAML, or JavaScript format to define the control flow of serverless functions and data flow between functions. A shared feature among all of these FC management systems is their use of a *Directed Acyclic Graph* (DAG) to represent the workflow.

Although the above-mentioned FC management systems are widely used to run scientific workflows [36], their languages to describe dependencies in the FC are mainly focused on data replication and distribution without diving into data types. In most cases, FC functions download data from cloud storage in a file format and upload the result of computing back to the storage. However, numerous modern applications are data-intensive in which many data items have relations with each other, represented in a graph format. Therefore, modern computing requires to execute many graph processing algorithms, such as *breadth-first search* to build indexes of crawled web pages, *depth-first search* to solve puzzles that have one solution, *shortest path* to guide a driver to travel from one location to another (*e.g.*, Google Maps), *cycle detection* to determine deadlocks in concurrent systems, or *graph coloring* to solve logic games (*e.g.*, Sudoku). Other examples try to determine a group of people that are strongly connected in social networks

ICPE '23 Companion, April 15–19, 2023, Coimbra, Portugal

ACM ISBN 979-8-4007-0072-9/23/04.
https://doi.org/10.1145/3578245.3585333

or schedule flight crews in airline companies. Such applications require adaptable graph data structures to represent complex systems and relationships. Thus, graph processing is a valuable tool for such applications and use cases, which often include recommendation systems [8], social network analysis [9], network optimization [18], fraud detection [24], and bioinformatics [31]. For this purpose, several graph processing platforms are introduced, such as *Apache Giraph* [1, 29], *Apache Flink* [7], *GraphMat* [42], and *GraphX* [49]. While these platforms support computation-intensive and unpredictable access patterns, most of them encounter scalability issues and cannot exploit all available serverless resources in federated FaaS, especially in cases when dealing with extremely massive and complex graphs. In general, they run on small-scaled clusters, usually tightly coupled, with a fixed number of compute nodes, which restricts them from adapting to the dynamic workloads with scaling in or out.

As a consequence, graph processing is limited by *two* perspectives. On the one hand, FC management systems, which can utilize serverless resources across the globe, offer limited support for graph processing. On the other hand, graph processing platforms support libraries for graph processing but with limited resource utilization. Therefore, *scalable serverless graph processing* requires careful investigation of the following research questions (RQs):

RQ1 **FC management systems to support graph processing:** To what extent can the existing serverless FC management systems and their languages to describe FCs be extended to offer support for large-scale graph processing?

RQ2 **Graph processing in federated FaaS:** To what extent can the existing graph processing platforms be migrated to serverless as FC to enlarge their scalability in federated FaaS?

To address these research questions, we will research and develop (prototype) solutions as part of the *Graph-Massivizer* project, funded by the Horizon Europe research and innovation program.

The remainder of the paper is organized in several sections. Section 2 surveys the state-of-the-art FC management systems and their support for graph processing in federated FaaS. We explore graph processing challenges and platforms with serverless approaches for graph processing in Section 3. Finally, Section 4 concludes the paper and presents our position for future work in serverless graph processing in federated FaaS with serverless workflows.

2 FC MANAGEMENT SYSTEMS IN FEDERATED FAAS

This section reviews the state-of-the-art of existing FC management systems and simulation frameworks for federated FaaS.

2.1 FC management systems of public cloud providers

AWS Step Functions is the FC management system of Amazon, which uses JSON to orchestrate functions into an FC using various constructs for parallelization, data distribution, as well as conditional branches. IBM Composer is another FC management system that utilizes JavaScript to develop the FC. Despite their offering for various constructs, both FC management systems utilize resources within the same cloud region of the same cloud provider without the possibility of running an FC across multiple cloud regions in federated FaaS, leading to limited scalability. Fig. 1a presents an example of an FC that is executed within a single cloud region. Google Workflows introduces similar constructs but uses YAML to determine the control and data flows. Compared to its competitors, it supports running functions via HTTP requests, which allows executing serverless functions on any cloud region, as illustrated in Fig. 1 c.

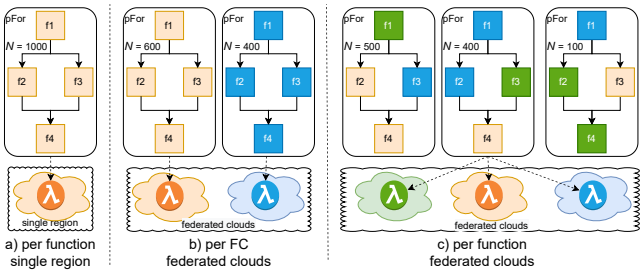


Figure 1: Various approaches for FC execution (a) within a single cloud region (*e.g.*, AWS Step Functions) (b) portable execution per FC (*e.g.*, Hyperflow), and (c) portable FC execution per function in federated clouds (xAFCL)

2.2 Open source FC management systems in a single cloud region

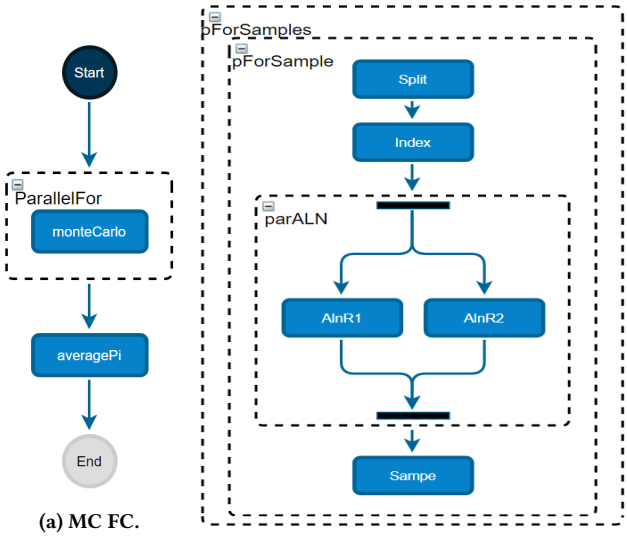
Apache OpenWhisk is one of the widely used open-source FC management systems, especially for evaluating scheduling algorithms [32, 44, 45, 48, 50]. Indeed, it relies on *Kubernetes* to manage function invokers and schedule on which container should execute each function. Hyperflow [28] is another open-source project that can port the entire FC from one to another FC management system, as shown in Fig. 1b.

2.3 FC management systems that offload computing on cloud

A few FC management systems are introduced that can run FCs across multiple commercial FaaS providers. However, their scalability is limited because they can offload functions of the FC on a single FaaS provider, which usually restricts the concurrency to a maximum of 1,000 functions. The MPSC [3] framework for Multi-Provider Serverless Computing balances a bag of tasks between the local edge devices and AWS Lambda and IBM Cloud Functions. Oscar [33] offloads functions to the cloud when the edge resources are overloaded. Hyperflow [28] supports running FCs either entirely on a specified region of AWS Lambda or Google Cloud Functions. However, neither of them considers the concurrency limitations of the FaaS providers.

2.4 FC management systems in federated FaaS

Ristov *et al.* introduced the xAFCL FC management system [37], which distributes an FC across various cloud regions of all top five FaaS providers (Fig. 1c). It uses the *Abstract Function Choreography*



(a) MC FC.

(b) BWA FC.

Figure 2: FCs in AFCL.

Language (AFCL), a YAML-based language, to describe complex control and data flows.

Fig. 2 depicts two examples of FCs represented as a graph with AFCL. Monte Carlo (MC) π approximation (Fig. 2a) consists of a sequence of two functions. The first sequence is a compound function, a parallel for loop, which scales with the number of `monteCarlo` base functions. After all instances of the `monteCarlo` base function finish, the FC runs `averagePi` function to collect results and returns the approximation of the π number. Despite its simple shape, the MC FC is intensively used in serverless research due to its parallelism [5, 6, 12, 13, 35, 39, 40].

The BWA FC (Burroughs-Wheeler Alignment) [25] (Fig. 2b) maps low-divergent sequences against a large reference genome from the Escherichia coli DNA. It is also a popular FC that is widely used in research [15, 37, 38, 51] due to the complex compute and data-bound requirements. BWA orchestrates five functions in a complex FC with two nested parallel loops and a parallel section.

While these two FC management systems, and many others, are commonly employed by researchers to assess different scheduling techniques, their main emphasis is typically on minimizing costs and maximizing performance, with rare attention paid to sustainability. Moreover, to the best of our knowledge, we could not find FCs for graph processing.

2.5 FaaS and FC simulation

The simfaas [47] emulator can simulate the runtime of serverless functions, the effects of cold start, and economics in FaaS platforms. Other frameworks, such as DFaaSCloud [20] and OpenDC Serverless [22], are simulators for FaaS platforms focus on functions' execution time. SimFaaS [27] simulates serverless functions with fixed memory setup in a single cloud region and models the average response time of functions, cold start, and concurrent instances of the serverless function. Recently, an advanced simulation framework *SimLess* [34] has been introduced as an extension of xAFCL [37]. It proposes a mathematical model to determine the overheads not only for a single function and their similar setups across federated FaaS, but for the entire FC. It should be noted that the aforementioned simulators/emulators allow for simulations of the performance, cost of functions, and FCs on a large scale, but still, they do not take sustainability into account.

3 GRAPH PROCESSING CHALLENGES AND SERVERLESS PLATFORMS

This section first presents several identified challenges for graph processing, and then describes several initial works for graph processing using serverless computing.

3.1 Graph processing challenges

Despite multiple FC management systems, which support the distribution of computing in federated FaaS, Lumsadaine *et al.* [26] identified the following challenges for parallel graph processing:

- Graph processing is data-driven because a graph comprises vertices and edges, which define how algorithms should perform the computations.
- Regardless of whether a graph is sparse or dense, its edges and vertices usually do not create embarrassingly parallel problems.
- Graphs have poor locality due to irregular characteristics.
- Graph processing generates a high data-access-to-computation ratio.

While various FC management systems introduce languages to specify parallelism, control flow, and data flow, they still do not consider the above-mentioned challenges. For example, suppose computing or data is skewed. In that case processing parallel loops will be inefficient because the makespan (*i.e.*, the time difference between the start and finish times) of the entire parallel loop would depend on the iteration that runs longest, while many other iterations would finish earlier. Additional skewness will be introduced by various overheads in federated FaaS, such as network latency or authentication, concurrency, FaaS, and session overheads [34].

3.2 Serverless graph processing platforms

Coimbra *et al.* [10] elaborated various approaches for graph processing, including single-machine and shared-memory parallel approaches, high-performance computing, and distributed graph-processing systems. In this section, we focus on serverless graph processing platforms.

Several initial works can be found in the literature, which used serverless computing to present (big) data processing systems [21, 23, 30]. However, none of them provides support for graph processing applications. Szalay *et al.* [43] employ resource partitioning and centralized cluster-level heuristics to schedule latency and throughput-sensitive serverless applications but do not take into account graph processing operations explicitly. Aslanpour *et al.* [4] present energy-aware scheduling for serverless edge computing. However, their solution is focused on edge devices and does not provide a graph processing platform that operates across the entire computing continuum nodes.

Heidari *et al.* [17] suggest a cost-effective auto-scaling method that adjusts the number and types of virtual machines according to the computation needs of convergent graph applications. However, this approach is incompatible with serverless paradigms since it does not consider the fundamental principles of serverless, *i.e.*, containerization and FaaS. The authors in [46] propose a serverless graph processing system called Graphless, which is implemented with AWS Lambda. Graphless allows graph processing functions to be deployed using push or pull operations on a set of predefined worker resources utilizing static and super-step schedulers. However, sustainability has not been considered in Graphless design. In addition, the authors demonstrate that the efficiency of Graphless can be degraded due to the variability of network characteristics under communication-intensive workloads.

Copik *et al.* [11] use three serverless functions that conduct graph processing (*i.e.*, PageRank, minimum spanning tree, and breadth-first search) as a part of the serverless benchmark suite named SeBS. With this approach, various serverless functions that conduct graph processing can be used to mimic their execution, including CPU and memory usage simulation [14].

While serverless functions that conduct graph processing are the necessary step towards scalable graph processing in federated FaaS, it is still the initial step. Running complex graph processing, such as PageRank, may take minutes, even for a small data set. On the other hand, parallelizing the steps and running this algorithm as an FC may significantly reduce the makespan, but more importantly, it can scale the graph's size, achieving both faster and scaled processing.

4 CONCLUSION AND FUTURE WORK

This paper designed two important research questions for large-scale graph processing, *i.e.*, (1) how existing serverless FC management systems can be extended to support graph processing and (2) how existing graph processing platforms can be migrated to a form of serverless FC management system. We surveyed state-of-the-art supports for graph processing using the existing FC management systems that can utilize serverless resources in federated FaaS. We also addressed the challenges of graph processing and showed that answering the aforementioned research questions requires further investigations. Therefore, we will research and develop (prototype) solutions that will address these questions in the context of the Graph-Massivizer project.

ACKNOWLEDGMENTS

This research received funding from:

- *Land Tirol*, under contract F.35499;
- *European High-Performance Computing Joint Undertaking*, under grant agreement 951745 (FF4EuroHPC project and CardioHPC experiment);
- *the Horizon Europe* research and innovation program of the European Union. Its grant management number is 101093202.

REFERENCES

[1] Apache. 2020. Apache Giraph. Retrieved 2023-02-12 from https://giraph.apache.org/

[2] Aitor Arjona, Pedro García López, Josep Sampé, Aleksander Slominski, and Lionel Villard. 2021. Triggerflow: Trigger-based orchestration of serverless workflows. *Future Generation Computer Systems* 124 (2021), 215–229. https://doi.org/10.1016/j.future.2021.06.004

[3] Austin Aske and Xinghui Zhao. 2018. Supporting Multi-Provider Serverless Computing on the Edge. In *Workshop Proceedings of the 47th International Conference on Parallel Processing (ICPP Workshops '18)*. ACM, Eugene, OR, USA. https://doi.org/10.1145/3229710.3229742

[4] Mohammad Sadegh Aslanpour, Adel N. Toosi, Muhammad Aamir Cheema, and Raj Gaire. 2022. Energy-Aware Resource Scheduling for Serverless Edge Computing. In *IEEE International Symposium on Cluster, Cloud and Internet Computing (CCGrid)*. 190–199. https://doi.org/10.1109/CCGrid54584.2022.00028

[5] Daniel Barcelona-Pons and Pedro García-López. 2021. Benchmarking parallelism in FaaS platforms. *Future Generation Computer Systems* 124 (2021), 268–284. https://doi.org/10.1016/j.future.2021.06.005

[6] Daniel Barcelona-Pons, Marc Sánchez-Artigas, Gerard París, Pierre Sutra, and Pedro García-López. 2019. On the FaaS Track: Building Stateful Distributed Applications with Serverless Architectures *(Middleware '19)*. ACM, Davis, CA, USA, 41–54.

[7] Paris Carbone, Asterios Katsifodimos, Stephan Ewen, Volker Markl, Seif Haridi, and Kostas Tzoumas. 2015. Apache flink: Stream and batch processing in a single engine. *The Bulletin of the Technical Committee on Data Engineering* (2015).

[8] Janneth Chicaiza and Priscila Valdiviezo-Diaz. 2021. A comprehensive survey of knowledge graph-based recommender systems: Technologies, development, and contributions. *Information* (2021).

[9] Avery Ching, Sergey Edunov, Maja Kabiljo, Dionysios Logothetis, and Sambavi Muthukrishnan. 2015. One trillion edges: Graph processing at facebook-scale. *Proceedings of the VLDB Endowment* (2015).

[10] Miguel E Coimbra, Alexandre P Francisco, and Luís Veiga. 2021. An analysis of the graph processing landscape. *journal of Big Data* 8, 1 (2021), 1–41.

[11] Marcin Copik, Grzegorz Kwasniewski, Maciej Besta, Michal Podstawski, and Torsten Hoefler. 2021. SeBS: A Serverless Benchmark Suite for Function-as-a-Service Computing. In *International Middleware Conference (Middleware '21)*. ACM, Québec city, Canada, 64–78. https://doi.org/10.1145/3464298.3476133

[12] Simon Eismann, Johannes Grohmann, Erwin van Eyk, Nikolas Herbst, and Samuel Kounev. 2020. Predicting the Costs of Serverless Workflows. In *Int. Conf. on Performance Engineering (ICPE)*. ACM, Canada, 265–276.

[13] Pedro García-López, Aleksander Slominski, Simon Shillaker, Michael Behrendt, and Barnard Metzler. 2020. Serverless End Game: Disaggregation enabling Transparency. *arXiv preprint arXiv:2006.01251* (2020).

[14] Ryan Hancock, Sreeharsha Udayashankar, Ali José Mashtizadeh, and Samer Al-Kiswany. 2022. OrcBench: A Representative Serverless Benchmark. In *2022 IEEE 15th International Conference on Cloud Computing (CLOUD)*.

[15] Nicholas Hazekamp, Nathaniel Kremer-Herman, Benjamin Tovar, Haiyan Meng, Olivia Choudhury, Scott Emrich, and Douglas Thain. 2018. Combining Static and Dynamic Storage Management for Data Intensive Scientific Workflows. *IEEE Trans. on Par. and Distr. Sys.* 29, 2 (2018), 338–350. https://doi.org/10.1109/TPDS.2017.2764897

[16] Michael T Heath. 2018. *Scientific Computing: An Introductory Survey, Revised Second Edition*. SIAM.

[17] Safiollah Heidari and Rajkumar Buyya. 2021. A Cost-Efficient Auto-Scaling Algorithm for Large-Scale Graph Processing in Cloud Environments with Heterogeneous Resources. *IEEE Transactions on Software Engineering* 47, 8 (2021), 1729–1741. https://doi.org/10.1109/TSE.2019.2934849

[18] Safiollah Heidari, Yogesh Simmhan, Rodrigo N. Calheiros, and Rajkumar Buyya. 2018. Scalable Graph Processing Frameworks: A Taxonomy and Open Challenges. *ACM Comput. Surv.* 51, 3 (jun 2018). https://doi.org/10.1145/3199523

[19] Sanghyun Hong, Abhinav Srivastava, William Shambrook, and Tudor Dumitras. 2018. Go Serverless: Securing Cloud via Serverless Design Patterns. In *10th USENIX Workshop on Hot Topics in Cloud Computing (HotCloud 18)*. USENIX Association, Boston, MA.

[20] Hongseok Jeon, Chunglae Cho, Seungjae Shin, and Seunghyun Yoon. 2019. A CloudSim-Extension for Simulating Distributed Functions-as-a-Service. In *International Conference on Parallel and Distributed Computing, Applications and Technologies (PDCAT)*. 386–391. https://doi.org/10.1109/PDCAT46702.2019.00076

[21] Eric Jonas, Qifan Pu, Shivaram Venkataraman, Ion Stoica, and Benjamin Recht. 2017. Occupy the Cloud: Distributed Computing for the 99%. In *Proceedings of the 2017 Symposium on Cloud Computing (SoCC '17)*. ACM, Santa Clara, California, 445–451. https://doi.org/10.1145/3127479.3128601

[22] S Jounaid. 2020. OpenDC Serverless: Design, Implementation and Evaluation of a FaaS Platform Simulator. Ph.D. thesis, Vrije Universiteit Amsterdam.

[23] Youngbin Kim and Jimmy Lin. 2018. Serverless data analytics with flint. In *2018 IEEE 11th International Conference on Cloud Computing (CLOUD)*. IEEE.

[24] Eren Kurshan and Hongda Shen. 2020. Graph computing for financial crime and fraud detection: Trends, challenges and outlook. *International Journal of Semantic Computing* (2020).

[25] Heng Li and Richard Durbin. 2010. Fast and accurate long-read alignment with Burrows–Wheeler transform. *Bioinformatics* 26, 5 (2010), 589–595.

[26] Andrew Lumsdaine, Douglas Gregor, Bruce Hendrickson, and Jonathan Berry. 2007. Challenges in parallel graph processing. *Parallel Processing Letters* 17, 01 (2007), 5–20.

[27] Nima Mahmoudi and Hamzeh Khazaei. 2021. SimFaaS: A Performance Simulator for Serverless Computing Platforms. In *Int. Conf. on Cloud Computing and Services Science (CLOSER '21)*. 1–11.

[28] Maciej Malawski, Adam Gajek, Adam Zima, Bartosz Balis, and Kamil Figiela. 2020. Serverless execution of scientific workflows: Experiments with HyperFlow, AWS Lambda and Google Cloud Functions. *Future Generation Computer Systems* 110 (2020), 502–514. https://doi.org/10.1016/j.future.2017.10.029

[29] Claudio Martella, Roman Shaposhnik, Dionysios Logothetis, and Steve Harenberg. 2015. *Practical graph analytics with apache giraph*. Vol. 1. Springer.

[30] Stefan Nastic, Thomas Rausch, Ognjen Scekic, Schahram Dustdar, Marjan Gusev, Bojana Koteska, Magdalena Kostoska, Boro Jakimovski, Sasko Ristov, and Radu Prodan. 2017. A Serverless Real-Time Data Analytics Platform for Edge Computing. *IEEE Internet Computing* 21, 4 (2017), 64–71. https://doi.org/10.1109/MIC.2017.2911430

[31] Pavlopoulos, Georgios A and Secrier, Maria and Moschopoulos, Charalampos N and Soldatos, Theodoros G and Kossida, Sophia and Aerts, Jan and Schneider, Reinhard and Bagos, Pantelis G. 2011. Using graph theory to analyze biological networks. *BioData mining* (2011).

[32] Thomas Rausch, Alexander Rashed, and Schahram Dustdar. 2021. Optimized container scheduling for data-intensive serverless edge computing. *Future Generation Computer Systems* 114 (2021), 259–271. https://doi.org/10.1016/j.future.2020.07.017

[33] Sebastián Risco, Germán Moltó, Diana M Naranjo, and Ignacio Blanquer. 2021. Serverless workflows for containerised applications in the cloud continuum. *Journal of Grid Computing* 19, 3 (2021), 1–18.

[34] Sashko Ristov, Mika Hautz, Christian Hollaus, and Radu Prodan. 2022. SimLess: Simulate Serverless Workflows and Their Twins and Siblings in Federated FaaS. Association for Computing Machinery.

[35] Sasko Ristov, Dragi Kimovski, and Thomas Fahringer. 2022. FaaScinating Resilience for Serverless Function Choreographies in Federated Clouds. *IEEE Transactions on Network and Service Management* (2022), 1–1. https://doi.org/10.1109/TNSM.2022.3162036

[36] Sasko Ristov, Stefan Pedratscher, and Thomas Fahringer. 2021. AFCL: An Abstract Function Choreography Language for serverless workflow specification. *Fut. Gen. Comp. Syst.* 114 (2021), 368 – 382.

[37] Sasko Ristov, Stefan Pedratscher, and Thomas Fahringer. 2021. xAFCL: Run Scalable Function Choreographies Across Multiple FaaS Systems. *IEEE Transactions on Services Computing* (2021), 1–1. https://doi.org/10.1109/TSC.2021.3128137

[38] Roland Mathá, Sasko Ristov, Thomas Fahringer, and Radu Prodan. 2020. Simplified Workflow Simulation on Clouds based on Computation and Communication Noisiness. *IEEE Transactions on Parallel and Distributed Systems* 31, 7 (2020), 1559–1574. https://doi.org/10.1109/TPDS.2020.2967662

[39] Josep Sampe, Pedro Garcia-Lopez, Marc Sanchez-Artigas, Gil Vernik, Pol Roca-Llaberia, and Aitor Arjona. 2021. Toward Multicloud Access Transparency in Serverless Computing. *IEEE Soft.* 38, 1 (2021), 68–74. https://doi.org/10.1109/MS.2020.3029994

[40] J. Sampe, M. Sanchez-Artigas, G. Vernik, I. Yehekzel, and P. Garcia-Lopez. 2021. Outsourcing Data Processing Jobs with Lithops. *IEEE Transactions on Cloud Computing* (Nov. 2021), 1–1. https://doi.org/10.1109/TCC.2021.3129000

[41] Josep Sampé, Gil Vernik, Marc Sánchez-Artigas, and Pedro García-López. 2018. Serverless Data Analytics in the IBM Cloud *(Middleware '18)*. ACM, Rennes, France, 1–8.

[42] Narayanan Sundaram, Nadathur Rajagopalan Satish, Md Mostofa Ali Patwary, Subramanya R Dulloor, Satya Gautam Vadlamudi, Dipankar Das, and Pradeep Dubey. 2015. Graphmat: High performance graph analytics made productive. *arXiv preprint arXiv:1503.07241* (2015).

[43] Márk Szalay, Péter Mátray, and László Toka. 2021. Real-time task scheduling in a FaaS cloud. In *2021 IEEE 14th International Conference on Cloud Computing (CLOUD)*. IEEE.

[44] Yang Tang and Junfeng Yang. 2020. Lambdata: Optimizing Serverless Computing by Making Data Intents Explicit. In *2020 IEEE 13th International Conference on Cloud Computing (CLOUD)*. 294–303. https://doi.org/10.1109/CLOUD49709.2020.00049

[45] Ali Tariq, Austin Pahl, Sharat Nimmagadda, Eric Rozner, and Siddharth Lanka. 2020. Sequoia: Enabling Quality-of-Service in Serverless Computing. In *Proceedings of the 11th ACM Symposium on Cloud Computing (SoCC '20)*. ACM, Virtual Event, USA, 311–327. https://doi.org/10.1145/3419111.3421306

[46] Lucian Toader, Alexandru Uta, Ahmed Musaafir, and Alexandru Iosup. 2019. Graphless: Toward serverless graph processing. In *2019 18th International Symposium on Parallel and Distributed Computing (ISPDC)*. IEEE.

[47] Erwin van Eyk. [n.d.]. SimFaaS. https://github.com/erwinvaneyk/simfaas. Accessed: 2021-10-22.

[48] Song Wu, Zhiheng Tao, Hao Fan, Zhuo Huang, Xinmin Zhang, Hai Jin, Chen Yu, and Chun Cao. 2021. Container lifecycle-aware scheduling for serverless computing. *Software: Practice and Experience* (2021).

[49] Reynold S Xin, Joseph E Gonzalez, Michael J Franklin, and Ion Stoica. 2013. Graphx: A resilient distributed graph system on spark. In *First international workshop on graph data management experiences and systems*. 1–6.

[50] Hanfei Yu, Hao Wang, Jian Li, and Seung-Jong Park. 2021. Harvesting Idle Resources in Serverless Computing via Reinforcement Learning. *arXiv preprint arXiv:2108.12717* (2021).

[51] Qimin Zhang, Nathaniel Kremer-Herman, Benjamin Tovar, and Douglas Thain. 2018. Reduction of Workflow Resource Consumption Using a Density-based Clustering Model. In *2018 IEEE/ACM Workflows in Support of Large-Scale Science (WORKS)*. 1–9. https://doi.org/10.1109/WORKS.2018.00006