Characterizing AFCL Serverless Scientific Workflows in Federated FaaS

Mika Hautz
University of Innsbruck
Computer Science Department
Innsbruck, Tyrol, Austria
mika.hautz@student.uibk.ac.at

Sashko Ristov
University of Innsbruck
Computer Science Department
Innsbruck, Tyrol, Austria
sashko.ristov@uibk.ac.at

Michael Felderer*
German Aerospace Center (DLR)
Institute for Software Technology
Cologne, Germany
michael.felderer@dlr.de

ABSTRACT

This paper introduces several, publicly available, serverless scientific workflows Montage, BWA, and Monte Carlo developed at a high level of abstraction using the Abstract Function Choreography Language (AFCL). Any individual function can run across federated FaaS comprising cloud regions of AWS and GCP. We present the support for composition with AFCL and execution with the xAFCL serverless workflow management system. For each AFCL workflow, we present implementation details, networking, and complexity. The evaluation of the presented serverless workflows shows that workflow functions download ephemeral data and run computation faster on AWS than on GCP. However, functions on GCP upload faster on the collocated storage.

CCS CONCEPTS

• **Computer systems organization → Cloud computing**.

KEYWORDS

Function-as-a-Service, federation, serverless, scientific workflows.

ACM Reference Format:
Mika Hautz, Sashko Ristov, and Michael Felderer. 2023. Characterizing AFCL Serverless Scientific Workflows in Federated FaaS. In *Proceedings of 9th International Workshop on Serverless Computing (WoSC '23).* ACM, New York, NY, USA, 6 pages. https://doi.org/10.1145/3631295.3631397

1 INTRODUCTION

Serverless computing is gaining more traction for running scientific applications in public cloud providers or on premises with various open source platforms [23]. Scientists usually code workflow tasks as serverless functions using Function-as-a-Service (FaaS) and orchestrate them in *serverless workflows* [13, 22, 24, 27]. Such scientific workflows include astrophysics (e.g., Montage [5]), bioinformatics (e.g., Burroughs-Wheeler Alignment (BWA) [19]), earthquake simulations (e.g., Cybershake [21]), and many more.

*Also with University of Cologne, Department of Mathematics and Computer Science.

Many cloud providers introduced serverless workflow management systems, such as AWS Step Functions or IBM Cloud Composer. However, these systems are mainly intended to be used for event-based and near real-time applications that often use other cloud services. Moreover, they are locked to run in the respective FaaS (AWS Lambda or IBM Cloud Functions) within a single cloud region, which restricts the required scalability for scientific workflows. Therefore, scientists are more focused to build open source serverless workflow management systems [2, 6, 11, 20, 24, 36, 38]. However, most of them support portable execution of the entire workflow, that is, to run the workflow on one or another provider.

In this paper, we develop and characterize several serverless workflows Montage, BWA, and Monte Carlo simulation. We selected these workflows because of their diversity in terms of 1) dynamic problem size that can be selected during runtime, e.g., based on the number of files stored in some cloud storage, 2) scalability vs. synchronization trade-off, and 3) various levels of computation, communication, and structure complexity. All workflows are composed in the Abstract Function Choreography Language (AFCL) [30] at a high level of abstraction. For each function of the workflow, a user can specify the function location (e.g., AWS Amazon Resource Name or URL) on any cloud provider. Unlike the native serverless workflow management systems of public cloud providers, which lock the users to run the workflows on their respective FaaS system, individual functions of AFCL workflows can be executed across any of the top cloud providers with the xAFCL serverless workflow management system [31]. These scientific workflows are publicly available, along with supporting tools to compose and run across *federated FaaS*, formed by various cloud regions of AWS and GCP. We also evaluated the workflows and determined that AWS provides better infrastructure for computation and downloading files from storage, while GCP's functions upload the files faster to the respective GCP cloud storage. Paper contributions include:

(1) publicly available[1] three scientific workflows AFCL-Montage, AFCL-BWA, and AFCL-MonteCarlo with details for their setup and execution in federated FaaS;

(2) workflow functions are coded in Python and can be deployed across AWS and GCP;

(3) characterization of workflow functions in terms of intermediary data transfer time and round trip time on AWS and GCP.

The remainder of the paper is organized into several sections. Section 2 presents the implementation, complexity, and networking of all evaluated AFCL workflows. The results of the evaluation and characterization are discussed in Section 3, while related work, workflow diversities, insights, and limitations are presented

[1]https://github.com/AFCLWorkflows

```
- function:
  name: "monteCarlo"
  type: "monteCarloType"
  dataIns:
  - name: "num_samples"
    type: "number"
    source: "ParallelFor1/num_samples"
  dataOuts:
  - name: "result"
    type: "number"
  properties:
  - name: "resource"
    value: "ARN|URL"
```

Figure 1: Base function specification.

in Section 4. Finally, conclusion and future work are presented in Section 5.

2 AFCL WORKFLOWS IMPLEMENTATION

In this section, we give a brief overview of portability with AFCL and implementation details of the AFCL serverless workflows AFCL-MonteCarlo, AFCL-Montage, and AFCL-BWA.

2.1 Portability with AFCL

General overview of AFCL. AFCL is a YAML-based language to orchestrate serverless functions in a serverless workflow with control and data flow constructs. The smallest computational task of an AFCL serverless workflow is the *base function*, which cannot be split into smaller tasks. For more complex serverless workflows, AFCL offers several *compound functions* to design the control and data flow. Most important constructs for serverless scientific workflows are `parallel` and `parallelFor`. The former allows parallelization of different tasks, while the latter represents data parallelism. AFCL workflows receive a JSON input, which is passed to base and compound functions with the data-flow.

Base function portability. Fig. 1 describes the main parts of the base functions that should be filled by the workflow developer. At the highest level of abstraction, workflow developers use *function types* to show *what* the function should do, without details for implementation or deployment. This includes the input data (`dataIns`), output data (`dataOuts`), and the unique function name within the workflow. With the function types, data, and control flow, the developer specifies the application-based information, without specifying the resources on which each function should run. This can be configured in the `resource` field under `properties`. Here, the developer can place the ARN (Amazon Resource Name) of the AWS Lambda, or e.g., URL of the function deployed on GCP. If developers want to run the presented function on another provider, they simply need to replace this value. The requirement of AFCL is that both functions should be of the same type, that is, to have the same data inputs, data outputs, and conduct the same work.

2.2 AFCL-MonteCarlo

Overview. Monte Carlo simulation is widely used by researchers as serverless workflow [3, 4, 8, 10, 33, 34]. The Monte Carlo approximation of π estimates its value by randomly generating points within a square and determining how many of these points fall within a quarter circle inscribed within the square. The more points are used, the closer the approximation gets to the actual value.

Implementation. The AFCL-MonteCarlo workflow consists of two different function types. The first function type is `monteCarlo`, which estimates the value of π. The second function type `averagePi` averages the results of the previous functions to get a final result of the estimate. While only a single instance of the `averagePi` function is needed, the number of `monteCarlo` functions can vary. Our version of the workflow consists of a parallel construct containing three sections, with each section containing a parallel loop with the `monteCarlo` function. Each of these three functions could be deployed to a different region, or even different provider to overcome the concurrency limit of the providers of 1,000 functions per region and provider, which often is much lower of maximum 100 functions [14, 31, 37]. In case fewer instances of `monteCarlo` are desired, the workflow could be simplified by only using a single parallel loop instead of the parallel construct.

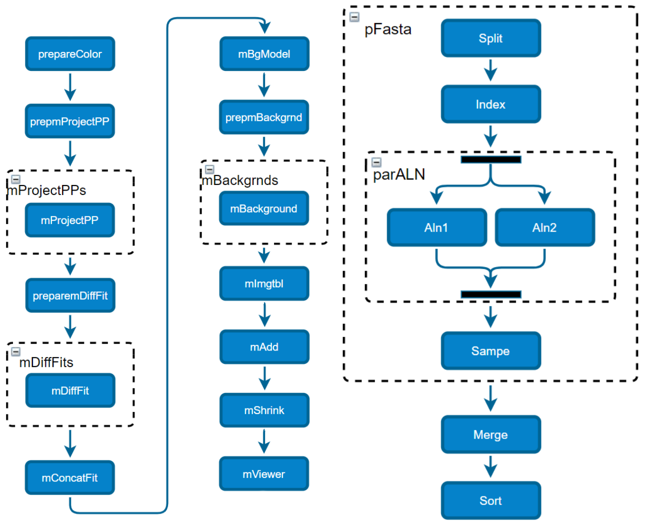
Complexity. The number of `MonteCarlo` functions is dynamically configurable with the workflow input. All functions exchange data by value, without accessing storage.

Output. The workflow outputs the result of `averagePi` (π estimation), which is the averaged number of all intermediate results produced by the `monteCarlo` functions.

2.3 AFCL-Montage

Overview. Montage [5] is an open source toolkit, which is created by the NASA/IPAC Infrared Science Archive. It can generate custom mosaics of the sky with input images in the Flexible Image Transport System (FITS) format. With a set of intermediary actions, Montage creates a final mosaic from the input images. Montage is a representative astronomy workflow widely used by many researchers [1, 2, 15, 24, 26].

Implementation. The structure of the AFCL-Montage workflow is presented in Fig. 2a). It comprises a sequence of 13 sequential functions including three `parallelFor` loops, each of which contains a single function. It is IO intensive and requires to transfer lots of data between the individual functions in order to create the final output mosaic. The first function `prepareColor` defines if a colored or gray image is created. The next step is to reproject the single input images to a pre-defined scale, which results in two new images per input image: a reprojected image and an *area*-image showing how much of the sky each part of the new image represents. This `mProjectPP` function can be orchestrated in a parallel loop, with each function getting exactly one image of the set of input images as an input. To determine the number of parallel iterations of the loop depending on how many input images are used, the `prepmProjectPP` function is invoked before the loop. After the first parallel loop, another helper function `prepmDiffFit` prepares the data for the next parallel loop that contains `mDiffFit`.



(a) AFCL-Montage. **(b) AFCL-BWA.**

Figure 2: AFCL serverless workflows.

It calculates the difference between a pair of reprojected images to determine all overlapping image pairs and fits a plane to the generated difference image. The output is then combined into a single table by the `mConcatFit` function. Afterwards, `mBgModel` determines a set of corrections to apply to each image in order to achieve the best global fit. The next step is to remove the background plane of the input FITS images, which is done in parallel again by `mBackground`, with `prepmBackground` determining the iterations for the parallel loop. `mImgtbl` creates an image metadata table and `mAdd` adds the reprojected images to form the mosaic in FITS format. Lastly, `mShrink` reduces the size of the mosaic by averaging blocks of pixels and `mViewer` generates the final JPEG image.

Complexity. We characterize the AFCL-Montage workflow with 0.25-degree, which creates 30 instances of the `mProjectPP` and `mBackground` functions, and 141 instances of the `mDiffFit` function. Particularly important for this workflow are the `mAdd` and `mShrink` functions, which transfer large ephemeral data of 116.05 MB and 51.58 MB.

Output. The result of AFCL-Montage is a combined mosaic of all input images with the corrected background. Based on the selection, it may be gray or in color.

2.4 AFCL-BWA

Overview. BWA is a scientific application that maps low-divergent sequences against a large reference genome from Escherichia coli DNA. With this workflow, medical persons may investigate the mutations of the bacteria, which shows whether the bacteria is curable or resistant to antibiotics. Many researchers use the BWA workflow [12, 25, 27, 28, 31, 39] due to its compute and data-bound requirements.

Implementation. AFCL-BWA (Fig. 2b) consists of seven functions, with the first five functions being nested in a parallel loop of four iterations. Each iteration processes a different sub-part of the reference genome to speed up the overall processing time by parallelizing the workload. The first function `split` is responsible for splitting the reference genome. It replaces all the bases outside of the split's window with null characters, keeping the amount of characters and order the same. In order to make the reference genome file better searchable, `index` uses the `bwa` executable to index the previously created reference genome split. Next, a parallel construct follows that runs two functions in parallel. Both functions `aln1` and `aln2` align the paired end reads of a DNA sample to the reference genome. Due to the DNA being read from both directions, two files need to be aligned which is the reason for requiring both of these functions. After the nested `parallel` construct, `sampe` uses both `SAI` files created by the `aln` functions to generate alignments in the `SAM` format. After each sub-part of the reference genome has been fully processed, `merge` uses the `samtools` executable to merge all `SAM` files of each reference genome split. Lastly, `sort` uses the `samtools` executable again to firstly sort the `SAM` file, followed by converting it to a binary representations that results in a `BAM` file and the indexed `BAM.BAI` file.

Complexity. All functions access storage to download a total of 1,042.56 MB or 109 files and upload a total of 181.7 MB or 39 files.

Output. The output files of the workflow can be analyzed with the Integrative Genomics Viewer (IGV) tool [32] to determine whether the E. coli sample is resistant to antibiotics. When the E. coli sample bears two distinct mutations within the hipA gene, it demonstrates an elevated occurrence of persisters. This heightened presence of persisters consequently empowers the bacteria with resistance against antibiotics [17]. In Fig. 3, the partial results from the IGV tool are depicted. The middle part of the image is trimmed out to show only the relevant parts. At the bottom, the reference genome with its bases is shown. Above, the series of gray bars each signifies a distinct read. When several readings show a difference from the expected genome, it becomes more probable that this difference is real and not just random variations. On the left side, all of the reads show that one base has transitioned from *G* to *A*. On the right side, one base has transformed from *A* to *C*. These specific alterations are the mutations responsible for endowing the E. coli bacteria with antibiotic resistance.

3 CHARACTERIZATION OF AFCL WORKFLOWS

This section characterizes AFCL-Montage, AFCL-MonteCarlo, and AFCL-BWA in terms of function round trip time, transfer times for download and upload, and overall makespan, for both cloud providers AWS and GCP.

3.1 Experiment setup

We deployed all evaluated workflows in two regions of AWS and GCP in London. Workflow functions accessed to their collocated storage, based on the recent work to move the functions close to data [35]. All functions of AFCL-Montage and AFCL-BWA, were deployed with 2 GB of memory, except for three functions that use

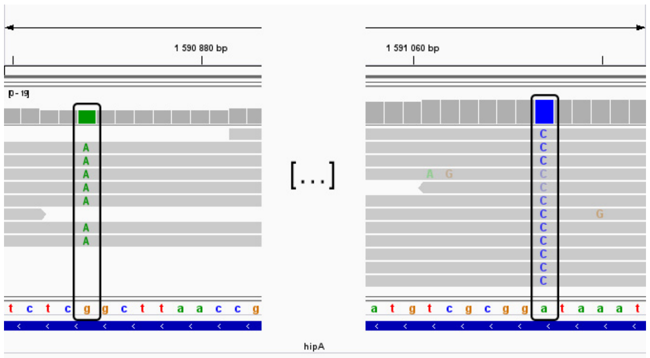


Figure 3: Two mutations in the *hipA* gene of the E. coli sample, determined after running AFCL-BWA.

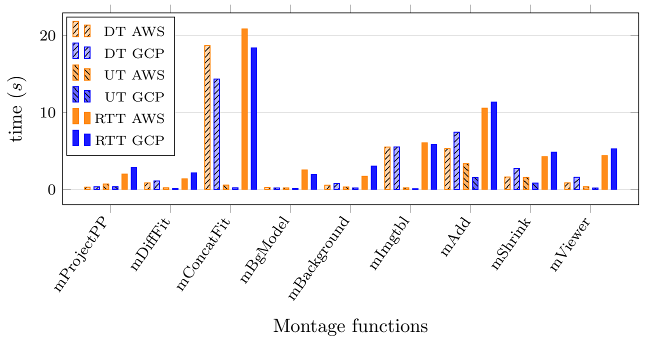


Figure 4: Characterization of AFCL-Montage functions' download, upload, and total round trip time while running the workflow on AWS and GCP London.

4 GB on GCP to avoid that the functions run out of memory. These functions are `mAdd`, `mImgtbl` and `mShrink` of the AFCL-Montage workflow. For AFCL-MonteCarlo, all functions use 128 MB because its functions do not require more memory.

The xAFCL enactment engine [31] was executed on a PC with an Intel i7-7700k CPU and 32 GB of RAM at the University of Innsbruck. We repeated the execution of each workflow setup six times and ignored the cold starts from the first execution, similar as other works [7, 27, 28].

3.2 Characterization of AFCL-Montage

Figure 4 characterizes AFCL-Montage on AWS and GCP in London. AFCL-Montage finishes on AWS in 65.67 s, thereby achieving a speedup of 1.2× compared to GCP. Surprisingly, functions `mConcatFit`, `mBgModel`, and `mImgtbl` run faster on GCP, but mainly due to faster download time, while the other functions run both computing and download time faster on AWS. Another interesting observation is that all uploads are faster on GCP.

3.3 Characterization of AFCL-MonteCarlo

We ran the MC_{10} AFCL-Workflow used for evaluation of SimLess [28]. That is, it runs $n = 9$ instances of the `monteCarlo` function. Figure 5 presents the characterization of AFCL-MonteCarlo

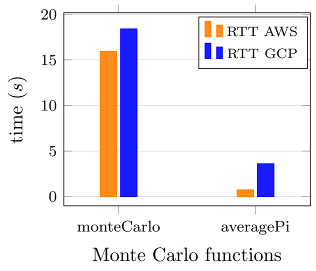


Figure 5: Characterization of AFCL-MonteCarlo functions round trip time on AWS and GCP London.

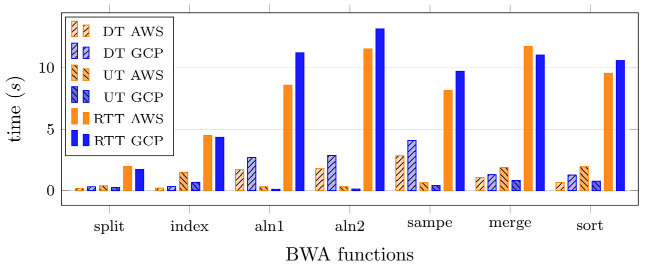


Figure 6: Characterization of AFCL-BWA functions' download, upload, and total round trip time while running the workflow on AWS and GCP London.

serverless workflow on AWS and GCP in London. AWS runs AFCL-MonteCarlo in 18.88 s, or 40.53 % faster than GCP. Both functions were faster on AWS.

3.4 Characterization of AFCL-BWA

Figure 6 characterizes AFCL-BWA on AWS and GCP in London. Overall, AFCL-BWA runs 49 s on AWS, or 12.92 % faster than on GCP (55.33 s). However, `split`, `index`, and `merge` run faster on GCP by 6.77 % on average. We observe that only a small part of the functions' round trip time is spent on data transfers, with the majority being attributed to computing.

All AFCL-BWA functions downloaded ephemeral data faster than their GCP counterparts, on average by remarkable 57 %. On the other side, all GCP functions upload ephemeral data by 51.84 % faster than AWS on average. Computationally, all AFCL-BWA functions on AWS, except `split` that does not run intensive computation, run the computation part faster than on GCP.

4 DISCUSSION

This section discusses the related work, diversity of implemented AFCL workflows, additional insights, and limitations.

4.1 Related work

Some of the AFCL workflows that were developed and evaluated in this paper were already developed for other serverless workflow management systems. Montage implementation is used with Hyperflow [24]. Some of the AFCL workflows were already used in federated FaaS for (1) simulation with SimLess [28], (2) spawn start evaluation [29], and (3) scheduling with FaaSt [27]. However, in this paper, we show implementation details how to compose the AFCL-workflows at the high-level of abstraction and run in federated FaaS. Larcher and Ristov [18] developed composite Backend-as-a-Service (BaaS) services by wrapping the BaaS services in functions and composed them in AFCL workflows.

4.2 Workflow diversity

The presented AFCL serverless workflows are very diverse and we believe that they are useful for the workflow and serverless communities to evaluate various optimization and simulation techniques with other serverless workflow management systems.

Dynamic workflow problem size. We exploited the AFCL option of `parallelFor` loops with dynamic loop iteration count and composed the AFCL workflows with this option. AFCL-MonteCarlo can change the problem size by increasing or decreasing the value of a single parameter `samples` in the workflow input in the *input_monteCarlo.json* file. This change does not require any change in the AFCL-MonteCarlo yaml file or in the code of the functions. Based on the scheduler decision for different function deployments of the `monteCarlo` function, the parameter `workers` should be adapted as well in the workflow input. This operation requires to adapt the workflow structure with more sections in the `parallel` construct, for which the transformation service of the FCEditor[2] can be used. On the other side, one can change the problem size of AFCL-Montage by adding FITS input files for the workflow with another size. For example, instead of input images with an angular size of 0.25 degrees, one could use inputs with 2.0 degrees, resulting in a larger field of view and therefore bigger files. This can be achieved without altering the workflow yaml file, but rather by solely changing the `input.json` file. Similar changes can be done to the other AFCL workflows to adapt the problem size.

Scalability vs. synchronization. Both the AFCL-Montage and AFCL-MonteCarlo workflows are embarrassingly parallel and can scale to hundreds of thousands of functions. In particular interest is the scalability - synchronization trade-off in workflows that have synchronization functions, such as the `merge` function in the AFCL-BWA workflow. We composed these workflows with the optimal trade-off between scaling the predecessor `parallelFor` loops and additional overhead in the synchronization task for the given workflow input. Additional mathematical models are needed to determine the optimal trade-off for arbitrary workflow inputs.

Computation, communication, and structure complexity. AFCL-MonteCarlo is a mainly computationally intensive workflow with zero communication cost, AFCL-BWA is mainly communication demanding, while AFCL-Montage is both computationally and communication intensive. AFCL-MonteCarlo and AFCL-Montage are implemented as a sequence of base and compound functions (`parallelFor`), while AFCL-BWA has a more complex structure, which includes nesting of base and compound functions into other compound functions.

[2]source code can be found on https://github.com/sashkoristov/FCeditor and publicly accessible on http://qe-fceditor.uibk.ac.at:8180/

4.3 Additional insights

Non-dominating FaaS providers. Overall, all workflows run faster on AWS due to faster download and computation time. On the other side, workflow functions upload files faster on GCP. Unlike the related work, such as Hyperflow [24] or Triggerflow [2], which can port the entire workflow from one to another FaaS provider, our AFCL workflows benefit from the xAFCL serverless workflow management system, which can run individual functions of the same workflow on different FaaS providers.

Data awareness. However, there is a pitfall by porting functions in Federated FaaS regarding the *data awareness*. Namely, recently, Eismann et al. [9] reported that more than 60 % of functions access storage. In our AFCL workflows, almost all functions download and upload intermediary data through storages of AWS or GCP. If a function is ported to another provider, then data access time changes, regardless if the storage is replaced or not. This insight requires additional research, which we set as our future work.

4.4 Limitations

Homogeneous memory. We used a diverse experiment setup in two FaaS providers in their regions in London. However, we restricted our evaluation with a single memory setup for almost all functions of each workflow, which was the minimum necessary memory to run all functions of the workflow. However, more optimal is to run each function with different memory in order to optimize its performance or cost, which we set as our future work.

Sample data vs. performance instability. We repeated each experiment for six times and averaged the values without the initial cold start. However, several researchers reported a huge deviance in the underlying infrastructures within a longer time period [16], as well as spawn start performance [14, 28, 29], especially for GCP and IBM.

5 CONCLUSION AND FUTURE WORK

In this paper, we presented three portable serverless scientific workflows AFCL-Montage, AFCL-MonteCarlo, and AFCL-BWA on two cloud providers AWS and GCP. All AFCL serverless workflows are publicly available including sources for their Python functions and AFCL codes of the abstract workflows. All serverless functions of the AFCL workflows can also be used by other serverless workflow management systems, by simply adapting the control and data flow for the new serverless workflow management system. The evaluation of the AFCL workflows showed that AWS functions run and download the files from AWS S3 faster than GCP. However, GCP functions upload files faster on GCP cloud storage.

We will extend our work in three directions:

(1) develop *memory-intensive* (e.g. satellite image processing), or *GPU-based* workflows (e.g. AI model training and federated learning) as AFCL serverless workflows and engineer them in a federated FaaS environment;

(2) use these characterizations in simulators, such as SimLess [28], to simulate execution of AFCL workflows with data awareness;

(3) develop a multi-objective scheduler that considers both performance and cost of AFCL serverless workflows in federated FaaS.

ACKNOWLEDGMENTS

This work is supported by Land Tirol, Austria, under F.35499.

REFERENCES

[1] Ali Al-Haboobi and Gabor Kecskemeti. 2021. Improving Existing WMS for Reduced Makespan of Workflows with Lambda. In *Euro-Par 2020: Parallel Processing Workshops: Euro-Par 2020 International Workshops, Warsaw, Poland, August 24–25, 2020*. Springer, 261–272.

[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] 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

[4] 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.

[5] G Bruce Berriman, Ewa Deelman, John C Good, Joseph C Jacob, Daniel S Katz, Carl Kesselman, Anastasia C Laity, Thomas A Prince, Gurmeet Singh, and Mei-Hu Su. 2004. Montage: a grid-enabled engine for delivering custom science-grade mosaics on demand. In *Optimizing scientific return for astronomy through information technologies*, Vol. 5493. SPIE, 221–232.

[6] Benjamin Carver, Jingyuan Zhang, Ao Wang, Ali Anwar, Panruo Wu, and Yue Cheng. 2020. Wukong: A Scalable and Locality-Enhanced Framework for Serverless Parallel Computing. In *ACM Symposium on Cloud Computing (SoCC '20)*. ACM, 1–15. https://doi.org/10.1145/3419111.3421286

[7] Henri Casanova, Rafael Ferreira da Silva, Ryan Tanaka, Suraj Pandey, Gautam Jethwani, William Koch, Spencer Albrecht, James Oeth, and Frédéric Suter. 2020. Developing accurate and scalable simulators of production workflow management systems with WRENCH. *Fut. Gen. Comp. Syst.* 112 (2020), 162 – 175.

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

[9] Simon Eismann, Joel Scheuner, Erwin van Eyk, Maximilian Schwinger, Johannes Grohmann, Nikolas Herbst, Cristina Abad, and Alexandru Iosup. 2021. Serverless Applications: Why, When, and How? *IEEE Software* 38, 1 (2021), 32–39. https://doi.org/10.1109/MS.2020.3023302

[10] 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).

[11] Jashwant Raj Gunasekaran, Prashanth Thinakaran, Nachiappan C. Nachiappan, Mahmut Taylan Kandemir, and Chita R. Das. 2020. Fifer: Tackling Resource Underutilization in the Serverless Era. In *Middleware*. ACM, Delft, Netherlands, 280–295. https://doi.org/10.1145/3423211.3425683

[12] 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

[13] Qingye Jiang, Young Choon Lee, and Albert Y Zomaya. 2017. Serverless execution of scientific workflows. In *International Conference on Service-Oriented Computing*. Springer, 706–721.

[14] Eric Jonas, Qifan Pu, Shivaram Venkataraman, Ion Stoica, and Benjamin Recht. 2017. Occupy the cloud: Distributed computing for the 99%. In *Symposium on Cloud Computing*. 445–451.

[15] Gideon Juve, Ann Chervenak, Ewa Deelman, Shishir Bharathi, Gaurang Mehta, and Karan Vahi. 2013. Characterizing and Profiling Scientific Workflows. *Fut. Gen. Comp. Syst.* 29, 3 (March 2013), 682–692. https://doi.org/10.1016/j.future.2012.08.015

[16] Daniel Kelly, Frank Glavin, and Enda Barrett. 2020. Serverless Computing: Behind the Scenes of Major Platforms. In *IEEE International Conference on Cloud Computing (CLOUD)*. 304–312. https://doi.org/10.1109/CLOUD49709.2020.00050

[17] Shaleen B. Korch and Thomas M. Hill. 2006. Ectopic Overexpression of Wild-Type and Mutant hipA Genes in Escherichia coli: Effects on Macromolecular Synthesis and Persister Formation. *Journal of Bacteriology* 188 (2006), 3826 – 3836. https://api.semanticscholar.org/CorpusID:22849731

[18] Thomas Larcher and Sashko Ristov. 2023. Scale Composite BaaS Services With AFCL Workflows. In *Workshops of The International Conference on High Performance Computing, Network, Storage, and Analysis (WORKS 2023)*. https://doi.org/10.1145/3624062.3624282

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

[20] Zhuozhao Li, Ryan Chard, Yadu Babuji, Ben Galewsky, Tyler J. Skluzacek, Kirill Nagaitsev, Anna Woodard, Ben Blaiszik, Josh Bryan, Daniel S. Katz, Ian Foster, and Kyle Chard. 2022. funcX: Federated Function as a Service for Science. *IEEE Trans. on Parallel and Distributed Systems* 33, 12 (2022), 4948–4963. https://doi.org/10.1109/TPDS.2022.3208767

[21] Philip Maechling, Ewa Deelman, Li Zhao, Robert Graves, Gaurang Mehta, Nitin Gupta, John Mehringer, Carl Kesselman, Scott Callaghan, David Okaya, et al. 2007. SCEC CyberShake workflows—automating probabilistic seismic hazard analysis calculations. *Workflows for e-Science: scientific workflows for grids* (2007), 143–163.

[22] Marcin Majewski, Maciej Pawlik, and Maciej Malawski. 2021. Algorithms for scheduling scientific workflows on serverless architecture. In *International Symposium on Cluster, Cloud and Internet Computing (CCGrid)*. IEEE, 782–789.

[23] Maciej Malawski and Bartosz Balis. 2022. Serverless Computing for Scientific Applications. *IEEE Internet Computing* 26, 4 (2022), 53–58. https://doi.org/10.1109/MIC.2022.3168810

[24] 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. *Fut. Gen. Comp. Sys.* 110 (2020), 502–514. https://doi.org/10.1016/j.future.2017.10.029

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

[26] Maciej Pawlik, Pawel Banach, and Maciej Malawski. 2020. Adaptation of workflow application scheduling algorithm to serverless infrastructure. In *Euro-Par 2019: Parallel Processing Workshops, Göttingen, Germany, August 26–30, 2019*. Springer, 345–356.

[27] Sashko Ristov and Philipp Gritsch. 2022. FaaSt: Optimize makespan of serverless workflows in federated commercial FaaS. In *International Conference on Cluster Computing (CLUSTER '22)*. IEEE, Heidelberg, Germany, 182–194. https://doi.org/10.1109/CLUSTER51413.2022.00032

[28] Sashko Ristov, Mika Hautz, Christian Hollaus, and Radu Prodan. 2022. SimLess: Simulate Serverless Workflows and Their Twins and Siblings in Federated FaaS. In *ACM Symposium on Cloud Computing (SoCC '22)*. ACM, San Francisco, CA, USA, 323–339. https://doi.org/10.1145/3542929.3563478

[29] Sashko Ristov, Christian Hollaus, and Mika Hautz. 2022. Colder Than the Warm Start and Warmer Than the Cold Start! Experience the Spawn Start in FaaS Providers. In *Workshop on Advanced Tools, Programming Languages, and PLatforms for Implementing and Evaluating Algorithms for Distributed Systems* (Salerno, Italy) *(ApPLIED '22)*. ACM, 35–39. https://doi.org/10.1145/3524053.3542751

[30] 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.

[31] Sasko Ristov, Stefan Pedratscher, and Thomas Fahringer. 2023. xAFCL: Run Scalable Function Choreographies Across Multiple FaaS Systems. *IEEE Trans. on Services Comp.* 16, 1 (2023), 711–723. https://doi.org/10.1109/TSC.2021.3128137

[32] James T. Robinson, Helga Thorvaldsdóttir, Wendy Winckler, Mitchell Guttman, Eric S Lander, Gad Getz, and Jill P Mesirov. 2011. Integrative genomics viewer. *Nature Biotechnology* 29, 1 (2011), 24–26. https://doi.org/10.1038/nbt.1754

[33] 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

[34] J. Sampe, M. Sanchez-Artigas, G. Vernik, I. Yehekzel, and P. Garcia-Lopez. 2023. Outsourcing Data Processing Jobs With Lithops. *IEEE Transactions on Cloud Computing* 11, 1 (2023), 1026–1037. https://doi.org/10.1109/TCC.2021.3129000

[35] Christopher Peter Smith, Anshul Jindal, Mohak Chadha, Michael Gerndt, and Shajulin Benedict. 2022. FaDO: FaaS Functions and Data Orchestrator for Multiple Serverless Edge-Cloud Clusters. In *International Conference on Fog and Edge Computing (ICFEC)*. 17–25. https://doi.org/10.1109/ICFEC54809.2022.00010

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

[37] Liang Wang, Mengyuan Li, Yinqian Zhang, Thomas Ristenpart, and Michael Swift. 2018. Peeking behind the Curtains of Serverless Platforms. In *USENIX Annual Technical Conference*. Boston, USA, 133–145.

[38] Minchen Yu, Tingjia Cao, Wei Wang, and Ruichuan Chen. 2023. Following the Data, Not the Function: Rethinking Function Orchestration in Serverless Computing. In *Symposium on Networked Systems Design and Implementation (NSDI 23)*. USENIX, Boston, MA, 1489–1504.

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