StoreLess: Serverless workflow scheduling with federated storage in sky computing

Sashko Ristov[1][0000−0003−1996−0098], Mika Hautz[1][0000−0001−7091−1238], Philipp Gritsch[1][0009−0008−9435−7364] Stefan Nastic[2][0000−0003−0410−6315], Radu Prodan[3][0000−0002−8247−5426], and Michael Felderer[4,5,1][0000−0003−3818−4442]

[1] University of Innsbruck, Austria
sashko.ristov@uibk.ac.at, m.hci.18@gmx.at, philipp.gritsch@uibk.ac.at
[2] TU Vienna, Austria
snastic@dsg.tuwien.ac.at
[3] University of Klagenfurt, Austria
radu.prodan@aau.at
[4] Institute of Software Technology, German Aerospace Center (DLR)
Michael.Felderer@dlr.de
[5] University of Cologne, Germany

Abstract. We observe irregular data transfer performance across federated serverless infrastructures (sometimes faster across providers than colocated), making the entire workflow scheduling even more challenging in federated FaaS and sky computing. This paper introduces StoreLess – a novel workflow scheduler and heuristic algorithm for serverless storage attachments that dynamically selects, provisions, and configures suitable function deployments and storage backends from the federated serverless infrastructure. StoreLess improves workflow execution time by up to 30 % by running cross-regional setup compared to the state-of-the-art.

Keywords: Federation · Serverless workflows · Scheduling · Storage

1 Introduction

Serverless workflows enable the composition of individual *functions-as-a-service (FaaS)* into a cohesive pipeline, allowing for seamless integration and coordination of various data processing steps. By modeling applications as serverless workflows, developers gain flexibility in managing dependencies, controlling data flow, and optimizing resource allocation for efficient execution. In serverless workflows, developers must explicitly communicate *intermediate data*, representing ephemeral input and output data transferred between steps.

Recently, *federated serverless* [6] and *sky computing* [11] have gained in popularity due to their numerous benefits to serverless workflows in terms of cost and performance. Existing serverful workflow management systems [3] optimize the workflow execution by proactively transferring the data to the virtual machines hosting the running tasks and aiming to hide communication. Unfortunately, this optimization is impossible for serverless functions that transfer intermediate data during runtime through cloud storage because their file system is

inaccessible to the workflow management system. Moreover, we observed irregular data transfer performance across federated cloud storage, sometimes faster across providers than colocated, making the workflow scheduling even more challenging (Section 2). However, existing schedulers colocate serverless workflows [9] or centralize the storage in a single region [6] without exploiting the potentially higher bandwidth to other cloud regions to optimize communication.

This paper introduces StoreLess, a novel workflow scheduler that delivers dynamic federated FaaS and storage to serverless workflows [6]. StoreLess aims to reduce the total workflow execution makespan by exploiting the function deployments and attached storage systems across regions and cloud providers (Section 3). We ran microbenchmarks to evaluate intra- and inter-region networking and drawn several surprising conclusions (Section 4). With this configuration, we evaluated StoreLess with two state-of-the-art approaches using two representative serverless workflows across six European and North American AWS and GCP regions. We reduced their makespan by up to 30 % compared to the colocated or federated FaaS setups (Section 5).

2 Motivational Study

We identified higher bandwidth across federated cloud regions compared to the intra-region. We implemented a simple `copyFile` function that downloads a file from AWS S3 and GCP Cloud Storage and uploads it back. We deployed the function `copyFile` in the AWS London (`AL`) region with 2 GB of RAM and configured it to upload a 100 MB file in `AL` S3, GCP London (`GL`), GCP Virginia (`GV`), and AWS Virginia (`AV`). We denote these setups as `ALAL`, `ALGL`, `ALGV`, and `ALAV` by concatenating the function deployment with the storage regions.

Observation: A function may upload data faster across regions than to its colocated storage. Counter-intuitively, Fig. 1 (left) shows that `ALGL` and `ALGV` performed respectively 68.26 % and 32.65 % faster, despite their geographical distance. We further narrowed down the benchmark for `ALAL` and `ALGL` while varying the file size from 25 MB to 100 MB, as presented in Fig. 1 (right). We determined that both regression functions intersect at 3.5 MB and estimated the upload time of 0.364 s.

3 StoreLess Workflow Scheduling

StoreLess targets serverless workflows $SW = (\mathbb{F}, \mathbb{D})$ composed of n serverless *functions* $\mathbb{F} = \bigcup_{i=1}^{n} f_i$ interconnected in a directed acyclic graph through a set of *data-flow dependencies* $\mathbb{D} = \{(f_i, f_j, d_{ij}, N_{ij}) \in \mathbb{F} \times \mathbb{F} \times \mathbb{R}^+ \times \mathbb{N}\}$. The function f_j can start only after its predecessor f_i finishes execution and generates *intermediate data* d_{ij} in N_{ij} files. StoreLess supports workflows for which d_{ij} is constant for a given workflow input data. StoreLess utilizes a *federated serverless infrastructure* with multiple *regions* $\mathbb{R} = \bigcup_{j=1}^{R} r_j$. We adopt the general

[6] https://github.com/xAFCL/StoreLess

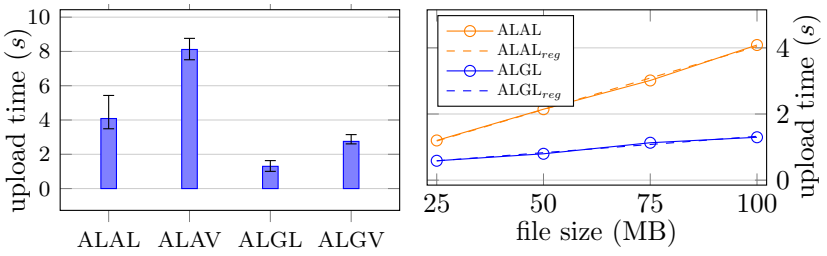


Fig. 1: Upload time of a 100 MB file (left) and different file sizes (right).

networking model [1] to estimate the *transfer time* of intermediate data d_{ij} with N_{ij} files from a function deployment region r_f to the storage region r_s:

$$TT(r_f, r_s, dir) = N_{ij} \cdot L\left(r_f, r_s, dir\right) + \frac{d_{ij}}{B\left(r_f, r_s, dir\right)}, \tag{1}$$

where $L\left(r_f, r_s, dir\right)$ and $B\left(r_f, r_s, dir\right)$ are the unidirectional *latency* and *bandwidth* between the function deployment and intermediate data storage regions, depending on the *transfer direction* $dir = \{\texttt{down}, \texttt{up}\}$ (download or upload).

We define a *function deployment* as a quadruplet $fd_{ir} = (f_i, r, r_s, RTT_{ir})$ that associates a function f_i with a computation region r and attached storage region r_s to upload the intermediate data. The expected RTT_{ir} of a function deployment has four components:

$$RTT_{ir} = ST_{ir} + FT_{ir} + DT_{ir} + UT_{ir} : \tag{2}$$

1. *Session time* ST_{ir} based on the SimLess' model [7];
2. *Function time* FT_{ir} necessary to run the computing part of the function;
3. *Download time* DT_{ir} to load intermediate data from predecessor functions;
4. *Upload time* UT_{ir} to upload output files to the attached storage region r_s.

We omit the cloud region s of the attached storage in the indexation because r_s affects only UT_{ir}.

We model the *workflow deployment* based on its scheduled functions and storage attachments. We model *earliest start time* $est\left(f_i\right)$ of a function f_i as the latest completion time of its predecessors $pred\left(f_i\right)$ (on their deployment region r'). Further on, we model *completion time* $ct\left(fd_{ir}\right)$ of a function f_i deployed in a region $r \in \mathbb{R}$ as the earliest start time plus its round-trip time RTT_{ir}. The *makespan* M_{SW} of a workflow SW is the completion time $ct\left(fd_{er}\right)$ of the end function f_e deployed in the region r:

$$est\left(f_i\right) = \max_{f_p \in pred(f_i)} \Big[ct\left(fd_{pr'}\right)\Big], \qquad ct\left(fd_{ir}\right) = est\left(f_i\right) + RTT_{ir}. \tag{3}$$

The StoreLess scheduling heuristic, formally presented in Algorithm 1 has three input parameters: a serverless workflow SW, a federated infrastructure

with R regions, and the deployment sets of all functions $\mathbb{FD}_i, \forall f_i \in \mathbb{F}$ (with the corresponding session, download, function, and upload time benchmark information). It returns a workflow deployment with the lowest makespan using a heterogeneous earliest finish time heuristic.

Algorithm 1: StoreLess scheduling algorithm

```
   Input  : SW = (𝔽, 𝔻), 𝔽 = ⋃_{i=1}^{n} f_i;                  // Serverless workflow
   Input  : ℝ = ∪_{r=1}^{R} r;                  // Federated infrastructure regions
   Input  : 𝔽𝔻_i = ⋃_{r=1}^{R} fd_ir, ∀f_i ∈ 𝔽;                // Function deployments
   Output: D_SW = {(f_i, sched(f_i)), ∀f_i ∈ 𝔽};          // Workflow deployment
 1 Function StoreLess (SW, ℝ, 𝔽𝔻):
 2     Rank ← B-Rank(𝔽);     // Order functions according to bottom rank
 3     D_SW ← ∅ ;        // Initialize workflow deployment with empty set
 4     for i ← 1 to n do                // Iterate over the ranked functions
 5         ct_min ← ∞;                                        // Initialization
 6         est_max ← est(Rank_i);              // calculate est based on Eq. 3
 7         for r ← 1 to R do          // Iterate over computational regions
 8             ST ← ST_{Rank_i r};   FT ← FT_{Rank_i r} ;        // load ST and FT
 9             DT ← DT_{Rank_i r} ; // Calculate download time based on Eq. 1
10             for r_s ← 1 to R do                 // Iterate storage regions
11                 UT ← UT_{Rank_i r} ;                       // Load upload time
12                 RTT_{Rank_i r} ← ST + FT + DT + UT ;      // based on Eq. 2
13                 ct ← ct(fd_{Rank_i r}) ;                   // based on Eq. 3
14                 if ct < ct_min then
15                     fd_min ← fd_{Rank_i r};      // save the fastest deployment
16                     ct_min ← ct;                // Save earliest finish time
17                 end
18             end
19         end
20         D_SW ← D_SW ∪ fd_min;                  // Add function deployment
21     end
22     return D_SW;                            // Return workflow deployment
23 return
```

Firstly, line 2 sorts all workflow functions according to their bottom rank [12], representing the critical path to the end of the workflow. The rationale of the ranking is to give a higher priority to the functions that have more dependent functions than others. Then, line 3 initializes the serverless workflow deployment plan with the empty set. Lines 4 to 21 iterate the ranked functions to deploy them to the most suitable region and dynamically attach storage. For each function (line 4), the algorithm first initializes the completion time and calculates the earliest start time of the function in lines 5 to 6, respectively. Further on, StoreLess traverses the function deployments of the function $Rank_i$ in lines 7 to 19. For each function deployment of the function, StoreLess loads ST

and FT (line 8) and estimates DT based on Eq. 1. These parameters are known since the regions of the input files, and the function deployment are known at this step. Further, the algorithm iterates over each storage region for their outputs (lines 10 to 18). Given that the function deployment and output storage regions are iterated (known), the algorithm estimates upload time UT based on Eq. 1. Afterward, it uses all computed and loaded parameters ST, DT, FT, and UT to compute RTT of the current function deployment in the region r with the current output storage r_s, based on Eq. 2. At each nested iteration, the algorithm saves the deployments with the earliest completion time (lines 14 to 17) and adds the fastest one to the workflow deployment in line 20. Line 22 returns the final serverless workflow deployment, ready for execution.

4 Implementation and infrastructure setup

To schedule data transfers, StoreLess relies on two variable types from Eq. 1: (i) cloud-specific (e.g., L, B) acquired using micro-benchmarks, and (ii) workflow or function-specific (e.g., N, d), incorporated directly in the workflow. We first measured upload and download time of files with 25 MB, 50 MB, 75 MB, and 100 MB between all evaluated regions within each provider. For this purpose, we developed a serverless function that uploads and downloads the file three times. We repeated the function five times and omitted the first execution to avoid the cold start. Further, we used the regression function to determine the bandwidth and latency in both upload and download directions.

Table 1 presents the parameter setup determined from the microbenchmark. Surprisingly, we observed several irregular data transfer parameters. First, the AWS download bandwidth B_{down} is larger than the upload bandwidth B_{up}, while latency follows the opposite pattern. For example, AWS S3 in London has the highest B_{down}, which is 3.15 times larger than B_{up}. However, GCP shows the opposite pattern, with B_{up} larger than B_{down}.

Table 1: Federated serverless infrastructure network model.

r	r_s	B_{up}	L_{up}	B_{down}	l_{down}
AL	AL	26 MB/s	0.231 s	82 MB/s	0.123 s
AL	GL	101 MB/s	0.351 s	93 MB/s	0.365 s
AF	AL	23 MB/s	0.453 s	85 MB/s	0.373 s
AF	GL	100 MB/s	0.396 s	151 MB/s	0.312 s
GL	GL	58 MB/s	0.109 s	43 MB/s	0.117 s
AL	GV	50 MB/s	0.877 s	58 MB/s	0.942 s

Surprisingly, the cross-provider bandwidth from AWS functions to GCP storage is significantly higher than the colocated setups for both providers. Notable is the AFAL setup with 100 MB/s upload compared to the colocated 26 MB/s for ALAL and 58 MB/s for GLGL, with comparable latency. The download bandwidth

for AFAL is even higher with 151 MB/s. ALGL shows a similar pattern, which is still worse than AFGL, despite the geographical closeness between functions and the storage. While, as expected, the ALGV upload and download latencies are higher than the colocated ALAL, the ALGV's bandwidth for upload is surprisingly nearly twice higher.

5 Experimental results

We use a double experimental strategy to enhance the completeness of the StoreLess evaluation: (i) *simulation* estimates the workflows' makespan using the model presented in Section 3; and (ii) *real testbed* validates the simulation by running the workflows in a federated infrastructure of AWS and GCP.

We selected two serverless workflows *Montage* [2] and *Burroughs-Wheeler Alignment (BWA)* [5] to challenge StoreLess. Both workflows are characterized in detail by Hautz *et al.* [4]. We used a federated testbed comprising several AWS and GCP regions across Europe and North America. We deployed the workflows in AWS London (AL), AWS Frankfurt (AF), GCP London (GL), and GCP Belgium (GB). We further selected four storage regions, AL, GL, AWS Virginia (AV), and GCP Virginia (GV). We run the scheduled workflows with the xAFCL serverless workflow management system [8] that can scatter the functions across federated FaaS by specifying their locations without updating the workflow structure.

Related work comparison. We evaluate StoreLess compared with two approaches. FADO [10], AWS Step Functions, or IBM Composer, *colocate the storage* in the same region running the workflow functions, assuming that network proximity minimizes data transfer times. FaaSt [6] supports *federated FaaS* but uses single storage due to a lack of support for storage federation. StoreLess federates all storage and all function regions.

5.1 Montage workflow

We present the Montage evaluation results with input first in AL, then in GL.

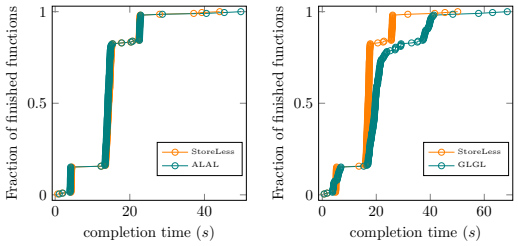
AL workflow input

Simulation. Fig. 2a presents the schedules of the function (r) and storage (r_s) of StoreLess, and FedFaaS approaches for an in-depth comparative analysis. We do not show the colocated schedules that place the functions and intermediate data in the same AL region with the workflow input. prepColor is the first function scheduled by StoreLess GLGL, with expected $RTT = 173.5\,\text{ms} + 109\,\text{ms}$. On the other side, FedFaaS cannot federate storage and decides among AFAL (0.9 s), ALAL (0.77 s), GBAL (1.1 s), and GLAL (0.93 s). Thus, FedFaaS schedules it to ALAL, although there is a faster mapping GLGL detected by StoreLess, expecting a faster execution by 0.49 s. prepmProject neither downloads nor uploads files but still accesses the AL storage to count the number of stored files. This function

runs fastest in AL (lower FT) with $RTT = 0.41$ s. However, StoreLess does not select ALAL as the first AWS function introducing a session time ST for a total of 0.96 s. Therefore, StoreLess schedules this function on GL again with RTT of 0.51 s. mProject - mImgtbl are all scheduled on ALAL by both StoreLess and FedFaaS as each instance downloads a small amount of data, insufficient to benefit from the cross-regional higher bandwidth of GCP regions.

Function	AL input region				GL input region			
	StoreLess		*FedFaaS*		*StoreLess*		*FedFaaS*	
	r	r_s	r	r_s	r	r_s	r	r_s
prepColor	GL	GL	AL	AL	GL	GL	GL	GL
prepmProject	GL	–	AL	–	GL	–	GL	–
mProject	AL	AL	AL	AL	AF	GL	AF	GL
prepmDiffFit	AL	AL	AL	AL	GL	GL	GL	GL
mDiffFit	AL	–	AL	–	AF	–	AF	–
mconcatFit	AL	AL	AL	AL	AL	AL	AF	GL
mBgModel	AL	AL	AL	AL	GL	GL	GL	GL
prepmBackgr	AL	–	AL	–	GL	–	GL	–
mBackground	AL	AL	AL	AL	AF	GL	AF	GL
mImgtbl	AL	AL	AL	AL	GL	AL	AL	GL
mAdd	AL	GL	AL	AL	GL	GL	GL	GL
mShrink	AF	GL	AL	AL	AF	GL	AF	GL
mViewer	AF	GL	AL	AL	AF	GL	AF	GL

(a) StoreLess schedules.



(b) AL input region. (c) GL input region.

Fig. 2: Schedules and executions with Montage and input in AL and GL.

However, the last three functions (mAdd, mSchrink, mViewer) differ significantly due to the large data transfers. mAdd deployed by StoreLess to ALGL dominates all schedules with an $RTT = 7.87s + 1.5s$ (UT), which is 25.37 % smaller than the colocated ALAL of the other schedulers, which need $UT = 4.68$ s, or 3.12× longer. mSchrink is the successor function deployed by StoreLess AFGL with the lowest download and computation times) and attached the GL storage (with the lowest upload time), minimizing the RTT by 35.8 % or reducing it by 1.8 s. mViewer is the last function deployed by StoreLess AFGL. It downloads the intermediate data from GL, with a minimum download and computation time of 4.06 s achieving an $RTT = 4.48$ s, which is 3.56 % higher than ALAL. Since FedFaaS scheduled mShrink to upload on AL storage, it allows a lower DT and FT of 4.01 s, including the lower time of 0.31 s to upload on AL, compared to 0.42 s to upload from AF to GL storage, as decided by StoreLess.

Real testbed validation. Fig. 2b presents the completion time distribution for executing Montage functions, which has an elevator shape with three large jumps representing the workflow parallel loops. We observe that the distribution of the completion times follows the estimated values of the StoreLess scheduler. For two reasons, the first two functions scheduled by StoreLess finish faster than the other two schedulers. First, StoreLess scheduled prepareColor in GL leading to 0.3 s including the shorter session time to GCP, while prepareColor ran 0.78 s, which includes the higher session time for AWS. mProjectPP ran 14.47 % slower with the StoreLess schedule because of the session time to AWS since that is the first function executed on AWS. Further on, the other functions until

`mImgtbl` followed the same pattern because of the same schedule. Finally, the main difference was observed for the last three functions. StoreLess reduced the RTT of the `mAdd` and `mShrink` functions 1.35× and 2.4×, respectively, achieving absolute reduction of 6.14 s for both functions. Still, this led to a slower $RTT = 0.33$ s or by 7 % for `mViewer`, as discussed before.

StoreLess advantage. StoreLess simulates the fastest makespan of 41.24 s, while both FedFaaS and colocated schedules expect 46.46 s, which is by 12.65 % slower. This improvement is not significant because it is achieved mainly for five functions, whose joint round trip times is 17.87 s and 23.09 s for the StoreLess and FedFaaS, respectively. Considering only these five functions, *FedFaaS* achieves a significant slowdown of 29.19 %. In real executions, the StoreLess schedule achieved similar improvement of 13.92 % compared to the other two schedules.

`GL` workflow input

Simulation. FedFaaS fixes the storage in `GL` and selects among the function deployments of the four regions `AL`, `AF`, `GL`, and `GB`, shown in Table 2a (right). StoreLess schedules the functions and storage in a similar way as for `AL` input region. FedFaaS generates almost the same schedule as StoreLess.

Real testbed validation. While StoreLess schedules with input in `AL` and `GL` follow the same pattern, the colocated schedule shows a significant delay, especially for functions in parallel loops. In other words, we achieved skewed elevations with positive gradients for the colocated parallel loops instead of verticals for two reasons: (i) GCP functions share the underlying infrastructure and starve for network bandwidth to the GCP storage, and (ii) simultaneous execution of multiple GCP functions increases their RTT, reported as concurrency overhead for massive function spawning [7]. The AWS functions from `AFGL` reported a negligible overhead compared to the colocated `GLGL`, especially for the largest parallel loop of 141 `mDifffit` instances (i.e., top of the second stair).

StoreLess advantage. StoreLess shows the main benefit for the workflow input in `GL` exhibiting 12.66 % improvement over colocation. StoreLess generates a similar schedule to FaaSt by attaching the GL storage to all functions except `mconcatFit`, estimated to run within 45.83 s, respectively 46 s. Surprisingly, the StoreLess real executions achieved even higher speedup of 1.3×, mainly from the functions deployed in `AF`. StoreLess ran within 51.19 s, or 1.12× longer, while the colocated schedule finished within 66.52 s, or 1.27× longer.

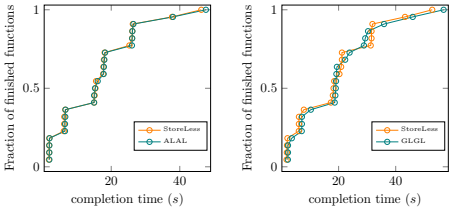
5.2 BWA workflow

The results for BWA are similar as for Montage and therefore we briefly discuss them. Since `Sampe` uploads 11.4 MB, the `AL` deployment needs 0.46 s upload time to `GL`, which is 0.2 s faster than storing it to `AL` storage. Since data is already

in GL, StoreLess schedules `Merge` as `AFGL` for downloading and uploading 45.6 MB. Notably, the estimated upload time for `ALGL` is 0.8 s, which is lower than the `AFGL`'s 0.85 s. The StoreLess schedule improvement is visible in the last three functions in Fig. 3b, i.e., running them 4 % faster than colocated.

Function	AL input region				GL input region			
	StoreLess		*FedFaaS*		*StoreLess*		*FedFaaS*	
	r	r_s	r	r_s	r	r_s	r	r_s
`split`	AL	AL	AL	AL	GL	GL	GL	GL
`index`	AL	AL	AL	AL	GL	GL	GL	GL
`aln1`	AL	AL	AL	AL	AF	GL	AF	GL
`aln2`	AL	AL	AL	AL	AF	GL	AF	GL
`sampe`	AL	GL	AL	AL	AF	GL	AF	GL
`merge`	AF	GL	AL	AL	AF	GL	AF	GL
`sort`	AL	GL	AL	AL	AF	GL	AF	GL

(a) BWA



(b) input in `AL`. (c) input in `AL`.

Fig. 3: Schedules and executions with BWA and input in `AL` and `GL`.

6 Related work

xAFCL [8] introduced a model to distribute bags of tasks as parallel loop iterations across multiple cloud regions. Similarly, FaDO [10] distributes files across multiple regions and colocates functions. However, both systems configured the loop iteration functions to access the colocated storage, assuming maximal performance. Recently, FaaSt [6] introduced a more granular distribution of workflow functions scattered in federated FaaS, which achieves better performance than xAFCL. However, the trade-off is that all workflow functions access single storage. StoreLess overcomes this weakness by allowing functions to dynamically select the storage regardless of the region where they are deployed.

Cheaper and faster sky computing. The high data transfer price is higher for cross-regional data transfers as it stems from external networking services to third-party providers. However, sky computing observations [13] reported cross-provider executions dominating the colocated ones in makespan and costs for ML pipelines thanks to the big data platforms overcoming the data transfer costs.

7 Conclusion and future work

We introduced StoreLess, a novel scheduler that minimizes the makespan of serverless workflows by exploiting that colocated functions and storage do not always provide the lowest transfer time. The main novelty introduced by StoreLess is a list-based orchestration heuristic to determine function deployments and storage attachments that maximize performance.

We plan to extend our work in two directions: 1. develop other serverless workflows for memory-intensive satellite image processing AI model training and

federated learning and apply StoreLess in federated serverless infrastructures; and 2. extend StoreLess to consider passing of intermediate data to workflow functions by-reference or by-value, as well as transformation of the control and data flow of the workflows to reduce data transfers.

Acknowledgement. This work received funding from Land Tirol, Austria (contract F.35499, TIM), KDT JU (grant agreement 101140216, MATISSE), Horizon Europe program (grant agreement 101093202, Graph-Massivizer), and Austrian Research Promotion Agency (grant agreement 903884, RapidREC).

References

1. Casanova, H., Giersch, A., Legrand, A., Quinson, M., Suter, F.: Versatile, scalable, and accurate simulation of distributed applications and platforms. Journal of Parallel and Distributed Computing **74**(10), 2899–2917 (2014)
2. Deelman, E., Singh, G., Livny, M., Berriman, B., Good, J.: The cost of doing science on the cloud: The montage example. In: ACM/IEEE Conference on Supercomputing. pp. 1–12 (2008)
3. Deelman, E., Vahi, K., Juve, G., et al.: Pegasus, a workflow management system for science automation. Future Generation Computer Systems **46**, 17–35 (2015)
4. Hautz, M., Ristov, S., Felderer, M.: Characterizing afcl serverless scientific workflows in federated faas. In: International Workshop on Serverless Computing. p. 24–29. WoSC '23, ACM, Bologna, Italy (2023)
5. Li, H., Durbin, R.: Fast and accurate long-read alignment with burrows–wheeler transform. Bioinformatics **26**(5), 589–595 (2010)
6. Ristov, S., Gritsch, P.: Faast: Optimize makespan of serverless workflows in federated commercial FaaS. In: Int. Conf. on Cluster Computing. p. 182–194. IEEE, Heidelberg, Germany (2022)
7. Ristov, S., Hautz, M., Hollaus, C., Prodan, R.: SimLess: Simulate serverless workflows and their twins and siblings in federated FaaS. In: Symposium on Cloud Computing. p. 323–339. ACM, San Francisco, CA, USA (Nov 2022)
8. Ristov, S., Pedratscher, S., Fahringer, T.: xAFCL: Run scalable function choreographies across multiple FaaS systems. IEEE Transactions on Services Computing **16**(1), 711–723 (2023)
9. Sethi, B., Addya, S.K., Bhutada, J., Ghosh, S.K.: Shipping code towards data in an inter-region serverless environment to leverage latency. J. Supercomput. **79**(10), 11585–11610 (Mar 2023)
10. Smith, C.P., Jindal, A., Chadha, M., Gerndt, M., Benedict, S.: Fado: Faas functions and data orchestrator for multiple serverless edge-cloud clusters. In: International Conference on Fog and Edge Computing (ICFEC). pp. 17–25 (2022)
11. Stoica, I., Shenker, S.: From cloud computing to sky computing. In: Hot Topics in Operating Systems. p. 26–32. HotOS '21, ACM, Ann Arbor, Michigan (2021)
12. Topcuoglu, H., Hariri, S., Min-You Wu: Performance-effective and low-complexity task scheduling for heterogeneous computing. IEEE Trans. on Par. and Distrib. Systems **13**(3), 260–274 (2002)
13. Yang, Z., Wu, Z., Luo, M., Chiang, W.L., Bhardwaj, R., Kwon, W., Zhuang, S., Luan, F.S., Mittal, G., Shenker, S., Stoica, I.: SkyPilot: An intercloud broker for sky computing. In: USENIX NSDI 23. pp. 437–455. Boston, MA (Apr 2023)