A Measurement Study on Serverless Workflow Services

Jinfeng Wen
Key Lab of High-Confidence Software Technology, MoE
Institute for Artificial Intelligence
Peking University, Beijing, China
jinfeng.wen@stu.pku.edu.cn

Yi Liu*
Key Lab of High-Confidence Software Technology, MoE
Peking University
Beijing, China
liuyi14@pku.edu.cn

***Abstract*—Major cloud providers increasingly roll out their serverless workflow services to orchestrate serverless functions, making it possible to construct complex applications effectively. A comprehensive study is necessary to help developers understand the pros and cons, and make better choices among these serverless workflow services. However, the characteristics of these serverless workflow services have not been systematically analyzed. To fill the knowledge gap, we conduct a comprehensive measurement study on four mainstream serverless workflow services, focusing on both features and the performance. First, we review their official documentation and extract their features from six dimensions, including programming model, state management, etc. Then, we compare their performance (i.e., the execution time of functions, execution time of workflows, orchestration overhead time of workflows) under various settings considering activity complexity and data-flow complexity of workflows, as well as function complexity of serverless functions. Our findings and implications could help developers and cloud providers improve their development efficiency and user experience.**

***Index Terms*—serverless workflow, cloud service, serverless computing**

I. Introduction

Serverless computing is a new paradigm in cloud computing, allowing developers to develop and deploy applications on cloud platforms without managing underlying infrastructure, e.g., load-balancing, auto-scaling, and operational monitoring [1], [2], [3], [4], [5]. Due to its significant advantages, serverless computing has been an increasingly hot topic in both academia [6], [7], [8] and industry [9], [10], [11], [12], whose market growth is expected to exceed $8 billion per year by 2021 [13]. In serverless computing, developers prototype an event-driven application as a set of interdependent functions (named *serverless functions*), each of which performs a single task [14]. To facilitate the coordination and composition among these functions, major cloud providers have rolled out serverless workflow services (e.g., AWS Step Functions [15]) to build complex, multi-step, stateful, and long-running application flows. Moreover, serverless workflow services allow developers to only specify the execution logic among functions, without having to deal with complex and error-prone communication and interactions among functions [6].

With the help of serverless workflow, complex applications (e.g., data processing pipeline and machine learning pipeline) can be constructed more efficiently [16]. Given the surging interest in serverless computing and the increasing dependence on serverless workflow, characterizing existing serverless workflow services is of great significance. On one hand, it can help developers understand the pros and cons of these services and thus make better choices among them based on various requirements. On the other hand, it can provide insightful implications for cloud providers to improve these services in a more targeted manner. However, to the best of our knowledge, the characteristics of these serverless workflow services have not been systematically analyzed.

To fill the knowledge gap, this paper presents the first comprehensive study on characterizing and comparing existing serverless workflow services. Specifically, we focus our analysis on four mainstream serverless workflow services, including AWS Step Functions (ASF), Azure Durable Functions (ADF) [17], Alibaba Serverless Workflow (ASW) [18], Google Cloud Composer (GCC) [19]. We first review their official documentation and characterize them in terms of six dimensions including orchestration way, data payload limit, parallelism support, etc. Then, we measure the performance (including the execution time of functions, execution time of workflows, orchestration overhead time of workflows, etc.) of these serverless workflow services under varied settings. The comparison is performed with two representative application scenarios, i.e., sequence applications and parallel applications, which refer to applications that can be prototyped as multiple functions executing in a sequence and parallel way, respectively. Sequence applications and parallel applications can be represented as sequence workflows and parallel workflows, respectively. More specifically, we focus on the following three research questions that we can provide insights for developers and cloud providers:

- **RQ1: How does activity complexity affect the performance?** We first compare the performance of these serverless workflow services under different levels of activity complexity [20] (i.e., the number of serverless functions contained in a workflow). We find that the execution time of workflows, execution time of functions, and orchestration overhead time of workflows all become longer for ASF, ADF, ASW, and GCC with the increase of activity com-

* Corresponding author, Yi Liu

978-1-6654-1681-8/21/$31.00 ©2021 IEEE
DOI 10.1109/ICWS53863.2021.00102

plexity in both sequence and parallel workflows, except that the orchestration overhead time of workflows of GCC has certain fluctuations in parallel workflows. Additionally, in sequence workflows, the execution time of workflows in ASF, ADF, and ASW are mainly generated by the execution time of functions, whereas GCC is the orchestration overhead time of workflows. In parallel workflows, when more functions are required, the execution time of functions gradually increases, and it determines the changing trend of the execution time of workflows.

- **RQ2: How does data-flow complexity affect the performance?** We then compare the performance of serverless workflow services under different levels of data-flow complexity [20] (i.e., the size of data payloads transferred among functions). We find that only under high data-flow complexity conditions ASF, ADF, and ASW will have a certain impact on the performance in sequence and parallel workflows, while GCC is affected by whether there is a payload or not.
- **RQ3: How does function complexity affect the performance?** We also compare the performance of these serverless workflow services under different levels of function complexity (i.e., the specified duration time of serverless functions). We find that the execution time of workflows and execution time of functions become longer for ASF, ADF, and ASW as function complexity increases in both sequence and parallel workflows, whereas there is no obvious impact on GCC. Besides, we find that the orchestration overhead time of workflows is less affected by function complexity.

Based on the derived findings, we have drawn insightful implications for both developers and cloud providers. Specific findings and implications are shown in Table I. We also open up the source code and detailed results[1] used in this study as an additional contribution to the research community for other researchers.

II. Feature Comparison

We select four mainstream serverless workflow services from public cloud platforms, including AWS Step Functions (ASF) (released December 1, 2016), Azure Durable Functions (ADF) (released May 7, 2018), Alibaba Serverless Workflow (ASW) (released July 2019), Google Cloud Composer (GCC) (released May 1, 2018). These services have relatively mature application practices and are more standardized rather than those of private cloud platforms. Through reviewing official documentation of these serverless workflow services, we compare their features from the following dimensions:

- Orchestration way: the workflow definition model and model definition language of serverless workflow services.
- Data payload limit: the size constraint of data payloads transmitted among serverless functions of a serverless workflow.
- Parallelism support: whether serverless workflow services support parallel multiple serverless functions.
- Execution time limit: the maximum execution time of workflows supported by serverless workflow services.
- Reusability: whether a serverless workflow can be used to a part (called sub-workflow) of another serverless workflow.
- Supported development language: the supported development languages for serverless workflow services.

Table II shows the feature comparison result as of Sep. 2020.

[1] https://github.com/WenJinfeng/ICWS21-ServerlessWorkflow

III. Methodology

We consider two representative scenarios, i.e., sequence applications and parallel applications, which can be represented as sequence workflows and parallel workflows, respectively. We present our methodology to measure and compare the performance of serverless workflow services under various settings.

Step 1: Determine performance metrics. In the first step, we will introduce the performance metrics of our study. Generally, the time of the whole process of workflow executions consists of the execution time of functions and the orchestration time of functions. Thus, the metrics related to them are considered, and the specific definitions are explained as follows. (i) ***totalTime*** is the total execution time to complete a workflow. (ii) ***funTime*** is the actual execution time of functions in a workflow. The calculation methods of *funTime* are different in different application scenarios. Specifically, in a sequence workflow, *funTime* is the sum of actual execution times of all functions in this workflow, whereas *funTime* in a parallel workflow is the time interval between the start time of the first function execution and the end time of the last function execution. (iii) ***overheadTime*** is the actual overhead time introduced in the orchestration process of a workflow. *overheadTime* may contain duration times of workflow start, function scheduling, data state transition, parallel branch and merge, etc. (iv) ***theo_overheadTime*** is the theoretical overhead time introduced in the orchestration process of a parallel workflow. Theoretically, executing paralleling multiple functions costs the time of the function with the longest execution time. The parallel workflow is theoretically a zero-overhead parallel composition. Removing the theoretical specified duration time of functions from the total time of workflows is the theoretical overhead time of workflows. Though comparing *overheadTime* with *theo_overheadTime*, we believe that some interesting findings can be found in our study. Note that the sum of *funTime* and *overheadTime* equals *totalTime* for a workflow.

Step 2: Set up the experiment. Most of our experiments were done from June 15 - August 20, 2020. In our study, without considering the cold start of spawning the function containers, serverless functions in workflows are in a *warm* state to avoid undesired startup latency. A common practice is to reuse launched containers by keeping them *warm* for a period of time. Each group of experiments is repeated multiple rounds to ensure the credibility of the results. We discord the result of the first measurement and keep the remaining results to evaluate the final performance of serverless workflow services. Additionally, we adopt the median value of remaining

TABLE I
SUMMARY OF FINDINGS AND IMPLICATIONS

Activity complexity	
Findings	**Implications**
F.1: The total execution time of workflows, execution time of functions, and orchestration overhead time of workflows become longer for ASF, ADF, ASW, and GCC with the increase of activity complexity in both sequence and parallel workflows, except that the orchestration overhead time of workflows of GCC has certain fluctuations in parallel workflows.	**I.1**: (i) For **developers**, we advise selecting the serverless workflow service with the best performance whether in both sequence and parallel workflows. (ii) For **developers**, considering the execution time of workflows, we advise that ADF is used in small-scale ($\leq$ 10) parallel workflow tasks, ASW is used in large scale (between 10 and 100) tasks, and ADF is used for larger-scale ($>$ 100) tasks.
F.2: In sequence workflows, the execution time of workflows in ASF, ADF, and ASW are mainly generated by the execution time of functions, whereas GCC is the orchestration overhead time of workflows.	**I.2**: The **cloud provider** of GCC can analyze the framework of workflow scheduling and improve significantly the efficiency of the workflow execution of GCC.
F.3: When more functions participate in sequence workflows, the orchestration overhead time of workflows gradually increases, and the orchestration overhead time determines the changing trend of the total time of sequence workflows.	**I.3**: For **developers**, considering the execution time and orchestration overhead time of workflows, we advise that ASF is used in activity-intensive sequence workflow tasks.
F.4: When more functions participate in parallel workflows, the execution time of functions gradually increases, and the execution time of functions determines the changing trend of the total time of parallel workflows.	**I.4**: **Cloud providers** of different serverless workflow services can further improve the efficiency of scheduling algorithms among functions to reduce the function execution overhead in parallel workflows.
Data-flow complexity	
Findings	**Implications**
F.5: Regarding the data payload limit, different serverless workflow services are different and do not have a uniform limit, i.e., ASF is 2^{18}B, ASW is 2^{15}B, GCC has an internal storage limit of 2^{15}B, and ADF does not have this restriction.	**I.5**: For **cloud providers** of different serverless workflow services, they can exchange technologies with each other to break through their respective implementation bottlenecks of data payload.
F.6: The performance of ASF, ADF, and ASW has little impact under low data payload conditions. Only under high data payload conditions will ASF, ADF, ASW have a certain impact, whereas GCC is affected by whether there is a data payload or not.	**I.6**: For **developers**, considering the execution time and orchestration overhead time of workflows, we advise that ASF is used in data-flow-intensive sequence (or parallel) workflow tasks, where payloads are less than 2^{18}B (or 2^{15}B).
F.7: ADF can pass a larger data payload (e.g., 2^{20}B). If developers use ASF, they can store data payloads in Amazon S3, then use the Amazon Resource Name (ARN) instead of raw data in serverless workflows.	**I.7**: For **developers**, when a workflow task needs to pass a larger data payload ($> 2^{18}$B in sequence workflows, or $> 2^{15}$B in parallel workflows), we advise using ADF, or ASF with the external storage.
Function complexity	
Findings	**Implications**
F.8: The function execution performance of ADF performs better than other serverless workflow services whether in sequence or parallel workflows.	**I.8**: **Cloud providers** of ASF, ASW, and GCC can improve the performance of serverless functions on their serverless computing platforms.
F.9: The orchestration overhead time of workflows is less affected by changes within serverless functions (function complexity), but more affected by changes in the workflow structure (activity complexity) or data payload (data-flow complexity).	**I.9**: For **developers**, considering the results of the execution time and orchestration overhead time of workflows, we advise that ASF (or ADF) is used in function-sensitive sequence (parallel) workflow tasks.

results to calculate *totalTime*, *funTime*, *overheadTime*, and *theo_overheadTime* of various serverless workflow services.

We leverage the *sleep* method to simulate functions with different complexity levels. In order to ensure function comparability, we try to write serverless functions in a consistent language. According to the serverless community survey, *Node.js* accounts for 62.9% of languages used for serverless development, and it is the most popular runtime overall [21]. In our study, ASF, ADF, and ASW adopt serverless functions with the *JavaScript* language. Because DAGs of GCC are written by a *Python* script, functions in GCC are achieved by the *Python* language. Each function is relatively simple and independently runs in a separate container. Moreover, we mainly focus on the comparison of different serverless workflow services rather than serverless functions. Thus, the results of these services are comparable. Additionally, ASF and ASW need to configure the function size in advance, thus we set it as 128MB of the memory uniformly. Regarding the region setting of serverless workflow services, ASF, ADF, and GCC are set to US-west uniformly, whereas ASW chooses Shanghai of China because it is only supported in Asia.

Step 3: Measure effects of activity complexity (RQ1). Activity complexity describes the number of functions that a workflow contains [20]. Considering both sequence and parallel workflows, we configure various numbers of functions in workflows. Because the maximum number of branches in parallel is limited to 100 in ASW, the function number of our study specifies as 2, 5, 10, 20, 40, 80, 100, and 120. Particularly, the experimental test about the function number with 120 is to verify the parallel limitations of ASF, ADF, and GCC, because their documentation does not explicitly indicate. Shahrad *et al.* [9] presented that the distribution of function execution times on Azure Functions [22] shows a sufficiently log-normal fit to the distribution of the average function execution time. We find that the probability that most function execution time is within one second, which illustrates that a majority of serverless functions run simple tasks. In experiments of activity complexity, we set all serverless functions to sleep for

TABLE II
A FEATURE COMPARISON IN FOUR SERVERLESS WORKFLOW SERVICES.

	AWS Step Functions	Azure Durable Functions	Alibaba Serverless Workflow	Google Cloud Composer
Orchestration way	State Machine / State Definition Language (JSON)	Orchestrator Function / In code	Flow / Flow Definition Language (JSON)	Directed Acyclic Graph (DAG) / In code
Data payload limit	256KB	Unknown	32KB	Unknown
Parallelism support	Yes	Yes	Yes	Unknown
Execution time limit	Standard type: 1 year Express: 5 minutes	Unlimited	1 year	Unlimited
Reusability	Yes	Yes	Yes	No
Supported development language	Java, .NET, Ruby, PHP, Python (Boto 3), JavaScript, Go, C++	C#, JavaScript, F#, PowerShell, Python	Java, Python, PHP, .NET, Go, JavaScript	Python 2, Python 3

one second.

Step 4: Measure effects of data-flow complexity (RQ2). Data-flow complexity reflects on data payload used in pre-and post-conditions of the function execution [20]. Considering both sequence and parallel workflows, we configure functions with various data payloads. Table II shows that the maximum data size between functions in ASW is 32KB (2^{15}B). To verify whether ASW can pass a larger data payload, we set data payloads with 0B, 2^5B, 2^{10}B, 2^{15}B, 2^{16}B. For most of sequence applications [16], we find that about five serverless functions can basically fulfill their requirements unless applications need to add additional functionalities. In our experiments, we set five serverless functions with the same functionality in parallel workflows.

Step 5: Measure effects of function complexity (RQ3). Function complexity reflects on the time required to execute a serverless function. Considering both sequence and parallel workflows with five same serverless functions, we configure various specified duration times of serverless functions. Shahrad *et al.* [9] mentioned that 96% of serverless functions take less than 60 seconds on average. Thus, we set the specified duration time of sleep functions as 50ms, 100ms, 1s, 10s, 20s, 40s, 60s, and 120s without data payloads.

IV. RESULTS

In this section, we show and discuss results under various levels of activity complexity, data-flow complexity, and function complexity considering both sequence and parallel application scenarios.

A. Activity Complexity (RQ1)

Activity complexity reflects on the number of serverless functions contained in a workflow.

1) Sequence application scenario: Fig. 1 represents *totalTime*, *funTime*, and *overheadTime* for various numbers of functions contained in sequence workflows for ASF, ADF, ASW, and GCC. The horizontal axis is the number of functions, and the vertical axis is the duration time in seconds. Each bar in Fig. 1 consists of *funTime* and *overheadTime* produced from the workflow with a fixed number of functions. Note that the sum of *funTime* and *overheadTime* equals *totalTime* for this workflow. The value next to the bar indicates the percentage of *funTime* to *totalTime*.

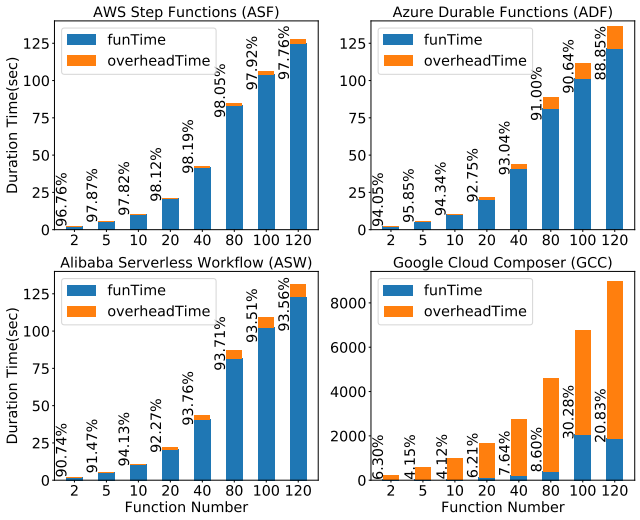

Fig. 1. The performance of various levels of activity complexity in sequence workflows.

For *totalTime* and *funTime* in Fig. 1, as more serverless functions are added to sequence workflows, *totalTime* and *funTime* of ASF, ADF, ASW, and GCC both increase. Undoubtedly, when the number of functions contained in sequence workflow increases, *funTime* will inevitably increase, thus *totalTime* increases. Generally speaking, five one-second functions have a *totalTime* of more than five seconds, ten one-second functions are more than ten seconds, etc. We find that *totalTime* of ASF, ADF, and ASW basically conforms to such a growing trend. Besides, *totalTime* of ASF, ADF, and ASW depends on *funTime*. Specifically, the percentage value of ASF fluctuates between 96.76% and 98.19%, ADF is between 91.00% and 95.85%, and ASW is between 90.74% and 94.13%. However, the percentage value of GCC is only between 4.12% and 8.60%, thus it illustrates that most of the time on GCC is spent on *overheadTime* rather than *funTime*. The main reason may be due to the environment setting itself.

For *overheadTime* in Fig. 1, we find that *overheadTime* of

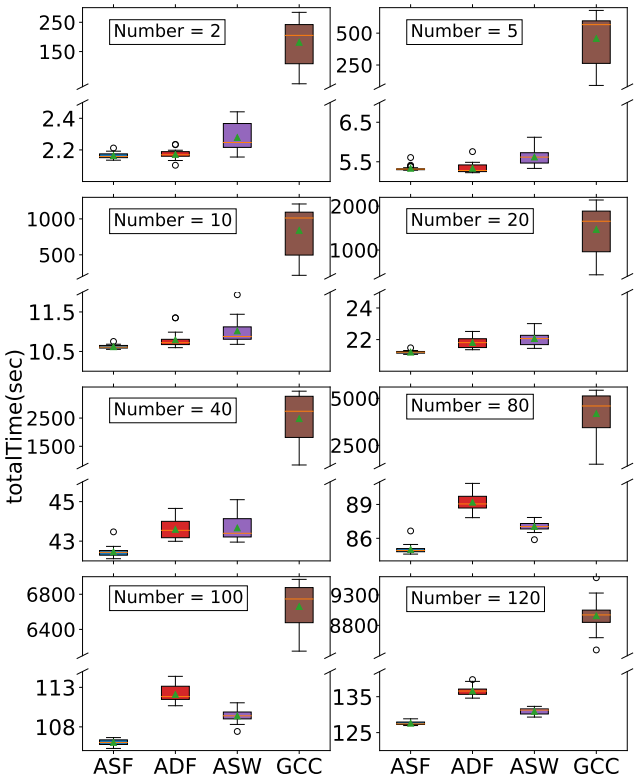

Fig. 2. The comparison of *totalTime* under various levels of activity complexity in sequence workflows.

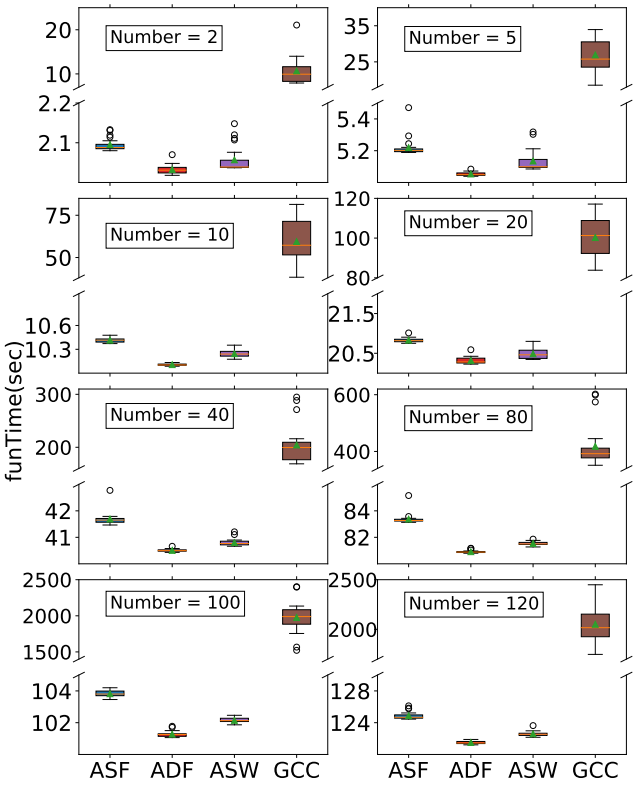

Fig. 3. The comparison of *funTime* under various levels of activity complexity in sequence workflows.

ASF, ADF, ASW, and GCC gets longer as more functions are added to sequence workflows. Thus, *the number of functions contained in sequence workflows will affect the orchestration overhead of workflows.*

To comprehensively compare the performance of ASF, ADF, ASW, and GCC, we display the statistical results of all measurements in the format of the box plot. Fig. 2 shows that the comparison of *totalTime* under varied numbers of functions for ASF, ADF, ASW, and GCC. We observe that ASF has the lowest and most stable *totalTime*, whereas GCC is the opposite. Additionally, when the number of functions contained in sequence workflows does not exceed 40, the overall result about *totalTime* of ADF is lower than that of ASW. However, when the number of functions increases (larger than 40), *totalTime* of ADF begins to exceed that of ASW. To explore which factors affect *totalTime*, we observe the distribution results of *funTime* and *overheadTime*. We find that *funTime* is longer than the theoretical execution time of functions in Fig. 3, where the origin point of the Y-axis on each sub-graph is the theoretical execution time of functions, i.e., 2s, 5s, 10s, 20s, 40s, 80s, 100s, 120s. Fig. 3 also shows that ADF has the lowest *funTime*, followed by ASW, ASF, and finally GCC. For the *overheadTime* comparison, its distribution is shown in our GitHub. We find that no matter how many functions are in sequence workflows, ASF often has the lowest *overheadTime*, and GCC is still the highest. In particular, for the changing trend of ADF and ASW, it is basically consistent with the comparison of *totalTime* in Fig. 2. Thus, *the changing trend of the total time of workflows in sequence workflow is mainly affected by the orchestration overhead time of workflows.* Reducing the orchestration overhead is vital for serverless workflow. Strategies about workflow start, state transition, and function scheduling need to be rethought to design by cloud providers.

2) Parallel application scenario: Fig. 4 shows that *totalTime*, *funTime*, and *overheadTime* of varied numbers of functions contained in parallel workflows for ASF, ADF, ASW, and GCC. As more serverless functions are added to parallel workflows, *totalTime* of ASF, ADF, ASW, and GCC is showing an increasing trend. From the percentage values (the ratio of *funTime* to *totalTime*) in ASF and ADF, we can observe that *totalTime* is mainly used for their *funTime*. Values in ASF range from 85.33% to 99.25%, whereas those of ADF are from 85.62% to 95.94%. Additionally, for ASW, when the number of functions is small, its *totalTime* is mainly used for function executions. However, when more functions are added to parallel workflows, its proportion values gradually decrease. It illustrates that *overheadTime* is increasing with the increase of the number of functions. On the contrary, for GCC, when the number of functions is small, its proportion is sufficiently low. It illustrates that the consumed time is longer

for *overheadTime* in GCC. When more functions participate in parallel workflows, *funTime* of GCC gradually increases. The possible reason is that there are many parallel functions and the execution scheduling between them is heavy.

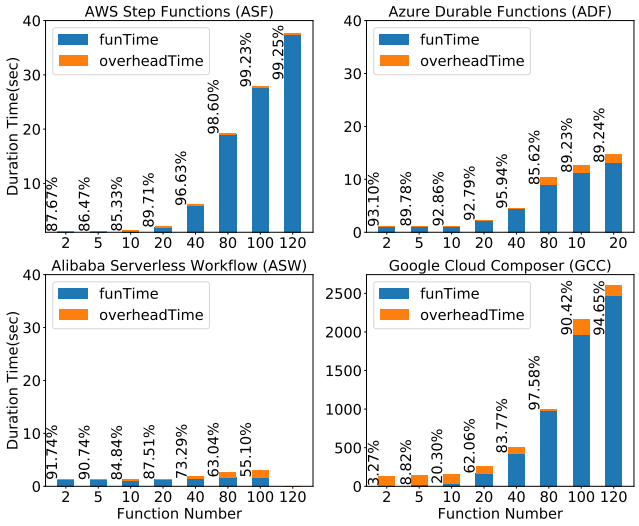

Fig. 4. The performance of various levels of activity complexity in parallel workflows.

In parallel workflows, theoretically, all serverless functions with the same task start and complete at the same time. Thus, excluding the execution time of a single function from *totalTime* is the theoretical orchestration overhead time (*theo_overheadTime*) of this workflow. Fig. 5 represents the comparison of *overheadTime* and *theo_overheadTime* under various numbers of functions contained in parallel workflows for ASF, ADF, ASW, and GCC. It shows that *theo_overheadTime* increases as more functions are added to parallel workflows. We find that *theo_overheadTime* is much larger than *overheadTime*. The value next to the bar indicates the percentage of *theo_overheadTime* that exceeds *overheadTime*. Specifically, for ASF, its value can arrive as high as 12862.54%, ADF is as high as 1832.45%, ASW is as high as 200.91%, and GCC is as high as 4020.73%. In addition, as the number of parallel functions increases, *overheadTime* becomes longer for ASF, ADF, and ASW. It takes a certain amount of time to process the branch and merge in parallel workflows. However, there are certain fluctuations in *overheadTime* of GCC, and fluctuations may be caused by its environment configuration.

Distributions of *totalTime*, *funTime*, and *overheadTime* are shown in our GitHub for various numbers of functions contained in parallel workflows. GCC has the longest *totalTime* in parallel experiments. When parallel functions with small-scale (less than or equal to 10), *totalTime* of ASF, ADF, and ASW is not much different, but the result of ADF is the lowest. When more functions (between 10 and 100) are paralleled into the workflow, ASW begins to show its advantages that have the *totalTime* result with lower and more stable compared to ASF and ADF. Since ASW has a limit of 100 for parallel tasks, no

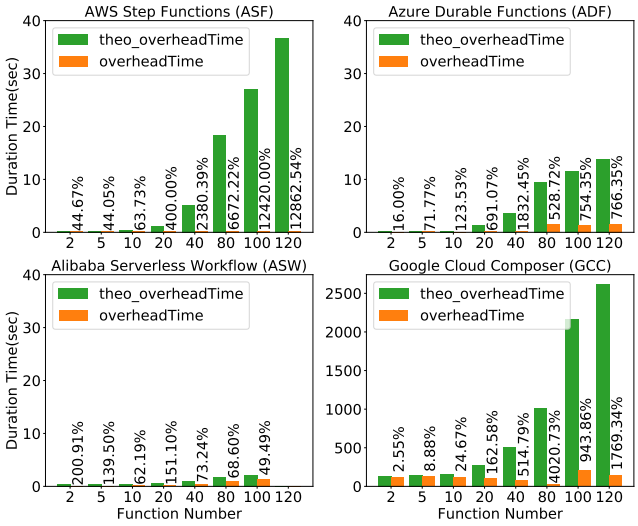

Fig. 5. The comparison of *overheadTime* and *theo_overheadTime* under various levels of activity complexity in parallel workflow.

more parallel functions can be executed. In the case of parallel functions with greater than 100, ADF can complete workflows in a shorter *totalTime*. Through observing the distribution of *funTime*, we find that the changing features of *funTime* are the same as *totalTime* distribution for ASF, ADF, and ASW. It illustrates that the changing trend of *totalTime* depends on *funTime*. Fig. 5 shows that the number of functions affects *overheadTime* of parallel workflows. Similarly, *overheadTime* distribution also shows such features. Moreover, when the number of functions does not exceed 40, ADF has the lowest *overheadTime*. As the number of functions increases from 40 to 120, ASF exhibits a lower *overheadTime* than ADF, ASW, and GCC. However, *overheadTime* values are relatively small for ASF, ADF, and ASW, and have little effect on their *totalTime*. Thus, in actual scenarios, the effect of *totalTime* in parallel workflows is usually considered.

See Findings F.1, F.2, F.3, F.4, and Implications I.1, I.2, I.3, I.4 in Table I.

B. Data-flow Complexity (RQ2)

Data complexity reflects on the sizes of data payloads passed among serverless functions in a workflow.

1) Sequence application scenario: Fig. 6 shows the performance of data payloads between 0B to 2^{16}B for ASF, ADF, ASW, and GCC. We add additional measurements for each serverless workflow service. (i) For ASF, the size limit of the data payload in a workflow is 256KB (2^{18}B). We conduct measurements about the data payload with 2^{18}B, and its *totalTime*, *funTime*, and *overheadTime* are respectively 6,599s, 5,683s, and 0.916s. In addition, when we add additional measurements that data payload is large than 2^{18}B, a validation error is detected, and it prompts the value at "input" failed to satisfy the constraint and must have a length less than or equal to 262,144 (i.e., 2^{18}B). This error illustrates that data payload

restriction described in the ASF documentation is consistent with the actual usage. (ii) For ADF, its documentation does not mention its size limit about data payload. In order to measure whether ADF supports a larger data payload, we conduct measurements of data payload with 2^{20}B, and its *totalTime*, *funTime*, and *overheadTime* are 27.613s, 6.688s, and 20.925s, respectively. (iii) For ASW, it exists the concept of local variable. When the payload is set as 2^{15}B, a failure occurs. It also verifies that the total size of the input, output and local variables of each step in ASW cannot exceed 32KB. To observe the impact of data payload, we add measurements of data payload with 2^{14}B to Fig. 6. (iv) For GCC, when data payload is set as 2^{16}B, a "mysql" error occurs that storing a message is bigger than 65,535 bytes. We check the environment resources of GCC and find that Cloud SQL is used to store metadata to minimize the possibility of data loss. Thus, experiments of data payload with 2^{16}B cannot be performed due to the storage limitation.

Fig. 6 shows that when data payload is less than or equal to 2^{10}B, *totalTime*, *funTime*, and *overheadTime* of ASF, ADF, and ASW have little effect. When data payload is greater than 2^{10}B, *totalTime* of ASF and ASW increases slightly. However, considering the results of ADF in data payload 2^{20}B, we find that *totalTime* of ADF increases significantly. Thus, we conclude that *the performance of ASF, ADF, and ASW has little impact under low data payload conditions. Only under high data payload conditions will ASF, ADF, and ASW have a certain impact.*

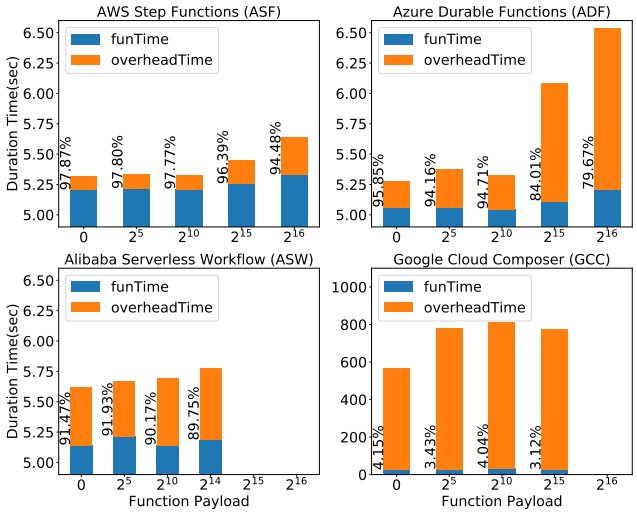

Fig. 6. The performance of various levels of data-flow complexity in sequence workflow.

To compare the result distribution of measurements, we show their respective box plots in our GitHub. For the comparison of *totalTime* under various data payloads in sequence workflows, in the low data payload range ($\leq 2^{10}$B), *totalTime* of ASF, ADF, and ASW is not much different, and *totalTime* of ASF and ADF is lower than ASW. In the high data payload range (between 2^{10}B and 2^{15}B), *totalTime* of ASF is the lowest. Considering previous analysis about data payload between 2^{15}B and 2^{18}B in Fig. 6, similarly, ASF is lower than ADF with regard to *totalTime*. However, whether in the low data payload or high data payload, *totalTime* of GCC is the highest. For the distribution of *funTime*, similar to Fig. 3, ADF has the lowest *funTime*, followed by ASW, ASF, and finally GCC. Specifically, when payloads are within 2^{10}B, values of ASF, ADF, and ASW maintain between 5s and 5.6s, while GCC is between 20s and 150s. Additionally, previous results of data payload between 2^{15}B and 2^{16}B in Fig. 6 also show that ASF is lower than ADF with regard to *overheadTime*. In this situation, ASF is the best choice. However, when data payload passing among functions grows to over 256KB, if developers still want to use ASF, advise using Amazon S3 to store the data, and pass the Amazon Resource Name (ARN) instead of raw data.

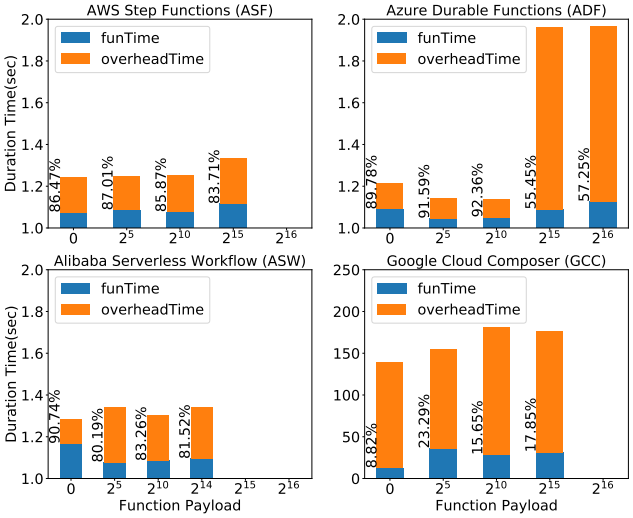

Fig. 7. The performance of various levels of data-flow complexity in parallel workflow.

2) Parallel application scenario: Fig. 7 represents the performance of various data payloads in parallel workflow for ASF, ADF, ASW, and GCC. First, we discuss the performance of *totalTime*. *totalTime* of ASF and ASW is basically not affected by low data payloads. However, transmit data payload with 2^{16}B into parallel workflows in ASF, and produce 5 times 2^{16}B data size to the workflow output. The high data output (larger than 2^{18}B) causes a failure of the workflow execution. For ASW, it has the data payload limit (32KB), and the merge of parallel functions also needs to be considered. From Fig. 7, we observe that when data payloads are within 2^{10}B in ADF, *totalTime* keeps stable. In the high data payload ($<$ 2^{10}B), *totalTime* of ADF increases. We also carry out parallel experiments with a data payload of 2^{20}B, where *totalTime* is 2,130s which is larger than data payload with 2^{16}B. For GCC, *totalTime* is affected by whether there is a payload or not. When there is a payload, *totalTime* will increase, but as the payload increases, it does not show a regular trend. Next, we discuss the performance of *funTime* and

overheadTime. *funTime* and *overheadTime* of ASF and ASW do not change much and are basically stable (maintaining acceptable fluctuations, e.g., 100ms). For ADF, under high data payloads, *overheadTime* increases greatly, and *funTime* does not change much. Thus, *overheadTime* of ADF under high data payloads is the main reason affecting *totalTime* change in parallel workflows. For GCC, *funTime* and *overheadTime* both increase in parallel workflows with the payload transmission. Thus, *only under high payload conditions will the performance of ASF, ADF, and SW has a certain impact, while GCC is affected by whether there is a payload or not.*

The result distributions of *totalTime*, *funTime*, and *overheadTime* are shown in our GitHub. Their distributions are similar. When data payload is set to be small ($\leq 2^{10}$B), *totalTime*, *funTime*, and *overheadTime* of ASF and ADF are low and relatively stable, whereas GCC is discrete and volatile. When data payloads are between 2^{10}B and 2^{15}B, *totalTime*, *funTime*, and *overheadTime* of ASF are the lowest. However, if developers want to pass into a large payload, only ADF, or ASF with the external storage can execute in parallel workflows.

See Findings I.5, I.6, I.7, and Implications I.5, I.6, I.7 in Table I.

C. Function Complexity (RQ3)

Function complexity reflects on the specified duration time of serverless functions contained in a workflow.

1) Sequence application scenario: Fig. 8 represents the performance of various specified duration times of functions in sequence workflows for ASF, ADF, ASW, and GCC. *totalTime* of ASF, ADF, and ASW increases as the specified duration time of functions gradually grows, whereas *totalTime* of GCC is not affected. Besides, there is no obvious trend in GCC for *funTime* and *overheadTime*. This situation is as described in "F.2" of Table I.

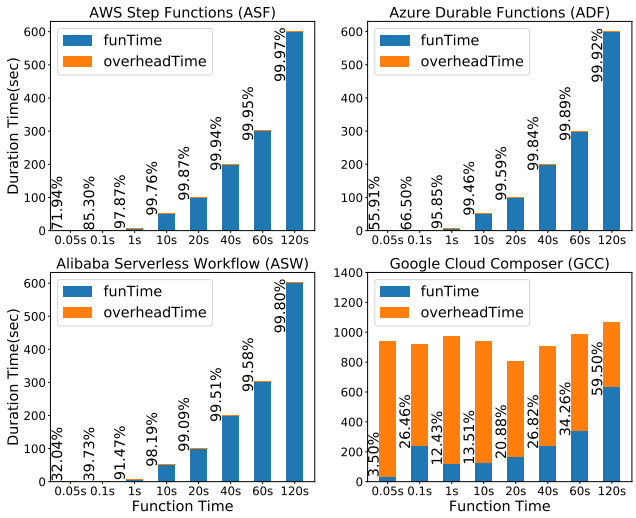

Fig. 8. The performance of various levels of function complexity in sequence workflow.

The changes of *overheadTime* are shown in Table III. When the number of functions contained in sequence workflows is fixed, *overheadTime* generally does not increase significantly for ASF, ADF, ASW, and they are roughly maintained within a certain range. Thus, we conclude that *changes within the function may not affect* overheadTime*, whereas changes between the workflow structure and data payload may have a certain impact on it.* However, the fluctuation of *overheadTime* of GCC is relatively large, ranging from 430s to 900s.

TABLE III
MEDIAN OF *overheadTime* (SECONDS) ABOUT THE SPECIFIED DURATION TIME OF FUNCTIONS IN SEQUENCE WORKFLOWS.

	0.05s	**0.1s**	**1s**	**10s**	**20s**	**40s**	**60s**	**120s**
ASF	0.188	0.121	0.113	0.119	0.127	0.120	0.150	0.157
ADF	0.246	0.272	0.219	0.271	0.408	0.320	0.339	0.507
ASW	0.738	1.036	0.479	0.926	0.922	0.978	1.264	1.207
GCC	903.720	676.320	852.360	810.600	637.020	663.540	647.280	433.320

Distribution figures about *totalTime*, *funTime*, and *overheadTime* are shown in our GitHub. We find that both ASF and ADF have a lower *totalTime* than ASW and GCC. Furthermore, the measurement results of ASF are more stable than ADF. For the *funTime* distribution, similar to Fig. 3, ADF has the lowest *funTime*, followed by ASW, ASF, and finally GCC. Regarding the distribution of *overheadTime*, ASF has the lowest result among all serverless workflow services overall.

2) Parallel application scenario: When the number of functions contained in parallel workflows is deterministic and the specified duration time of functions increases, *totalTime* and *funTime* must increase. However, *totalTime* and *funTime* of GCC do not increase with the increase of the specified duration time of functions in parallel workflows. This figure is shown in our GitHub. To observe the changes in *overheadTime* more intuitively, Fig. 9 shows the comparison for *overheadTime* and *theo_overheadTime*. We find that *theo_overheadTime* is larger than *overheadTime*. We also find that *overheadTime* and *theo_overheadTime* for ASF, ADF, ASW, and GCC do not change significantly with the increase of the specified duration time of functions, and they basically fluctuate within a certain range. ASF, ADF, and ASW are below 0.5s, while GCC is below 200s. At the same time, Fig. 9 shows that *overheadTime* of ASF is relatively stable, whereas ADF, ASW, and GCC have certain fluctuations.

The distributions of *totalTime*, *funTime*, and *overheadTime* are similar and they are shown in our GitHub. We find that the results of ADF are the lowest in terms of *totalTime*, *funTime*, and *overheadTime*, whereas GCC is the highest. However, as far as the stability of the result is concerned, ASF is the best. For ASW, its results are relatively unstable compared with ADF, and there are more outliers.

See Findings F.8, F.9, and Implications I.8, I.9 in Table I.

V. DISCUSSION

For verifying our findings, we conduct experiments of two real-world serverless application workloads, i.e., KMeans and MapReduce. Then, we discuss some limitations of our study.

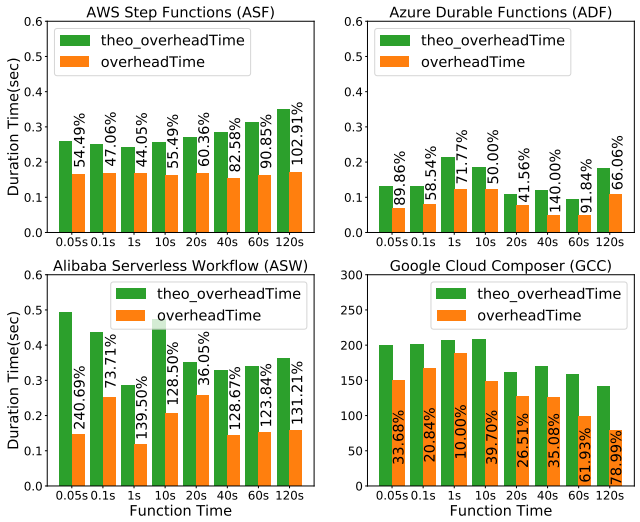

Fig. 9. The comparison of *overheadTime* and *theo_overheadTime* under various levels of function complexity in parallel workflow.

(1) KMeans application is implemented in a sequence workflow, accomplishing the clustering functionality for point sets with three-dimensional space. First, use a serverless function to generate 1500 points, because the data payload limit of ASW cannot generate the data of 2000 points. Second, initialize centroid points randomly. For the *KMeans* algorithm, the K-value of clustering needs to be given in advance. We adopt *Elbow Method* to determine K as 8. Next, based on the point set and centroid points, perform the clustering functionality of *KMeans*. Finally, output and show the clustering result.

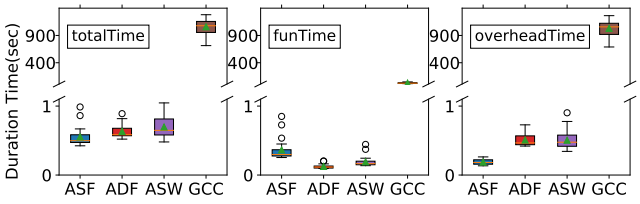

Fig. 10. The performance of the *KMeans* application.

Fig. 10 represents the comparison of *totalTime*, *funTime*, and *overheadTime* of the *KMeans* application for ASF, ADF, ASW, and GCC. ASF shows the shortest *totalTime* and *overheadTime*, and ADF has the shortest *funTime* (**F.8 in Table I**). This is consistent with the implication **I.3 in Table I**. We also find the same finding that the changing trend of *totalTime* in sequence workflow is mainly affected by *overheadTime* (**F.3 in Table I**) because *funTime* costs the relatively low and stable time in this *KMeans* application. In terms of data-flow complexity about data payloads, the previous conclusion (**I.6 in Table I**) is that when data payload is less than 2^{18}B, ASF is advised to use. In the *KMeans* application, the data payload size is within 2^{18}B, and the performance of ASF is the best considering *totalTime*, *overheadTime*.

(2) MapReduce application is implemented in a parallel workflow and is a workflow solution example [23]. The application goal is to generate a batch of data and process them. Count the number of occurrences of various data leveraging *MapReduce* processing frame mode.

I.1 in Table I presents that ADF is used in small-scale activity-intensive parallel workflows. Fig. 11 also shows that ADF has a relatively short *totalTime*. However, the results from *totalTime* and *overheadTime* of ASF are more stable than ADF. In the *MapReduce* application, there is a certain data payload to be transmitted. In the presence of data payload, the previous conclusion is that the ASF is more suitable when data payloads are less than 2^{15}B in parallel workflows (**I.6 in Table I**). Thus, the results of ASF show relatively satisfactory *totalTime* and *overheadTime*.

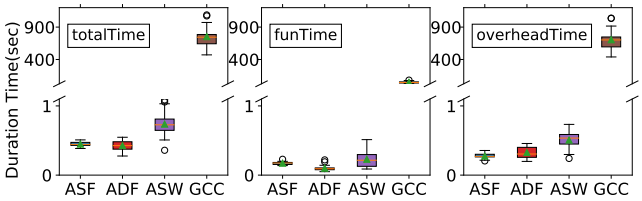

Fig. 11. The performance of the *MapReduce* application.

Limitations. We discuss the limitations of our study. (i) **Selection of application scenarios.** Our study is based on sequence and parallel workflows. We may ignore other complex structures, e.g., choice, missing valuable insights with regard to the structure complexity of workflows. In future work, we plan to extend our study to diversify workflow structure to further obtain interesting findings. (ii) **Experiments of GCC.** In our study, the results of GCC fluctuate greatly, and we suppose that it may be related to its environmental setting. To minimize this impact, we repeat several measurements. From the perspective of serverless computing, we suppose that functions performed in DAGs of GCC are not serverless, i.e., GCC is not designed for orchestrating serverless functions. Communication ability between non-serverless functions may be unstable on the cloud platform, thus orchestration overhead dominates most execution time.

VI. Related Work

In this section, we summarize the related work of serverless computing and serverless workflow service.

Serverless computing is going to attract more attention. Wang *et al.* [24] conducted a measurement study to characterize architectures, performance, and resource management of three serverless computing platforms. Maissen *et al.* [25] designed a benchmark suite named *FaaSDOM* to facilitate the performance testing of serverless computing platforms. These studies are mainly from the perspective of serverless functions. Different from them, our work focuses on the comparison of mainstream serverless workflow services under the serverless computing paradigm.

To facilitate the coordination of serverless functions on serverless computing platforms, the authors [26] compared serverless workflow services from their architectures and workflow models in 2018. However, new information has been updated on their official website. For example, Azure Durable Functions currently has supported *JavaScript* and *Python* languages. Akkus *et al.* [6] ran an image processing pipeline using serverless workflow services. They found that the total execution time of workflows is significantly more than the actual time required for function execution. As confirmed by our study, a part of the time will be used for orchestration scheduling between functions. Recently, the authors [27] presented *Triggerflow*, which is an extensible trigger-based orchestration architecture integrating various workflow models, e.g., *State Machines*, *Directed Acyclic Graphs*, and *Workflow as code*. It can be seen that there is a growing interest in serverless workflows. Furthermore, we notice that serverless workflow services are evolving quickly. Nevertheless, our findings and implications serve as a snapshot of orchestration mechanisms, provide performance baselines and suggestions for developers to implement reliable and satisfying applications and help cloud providers improve service efficiency.

VII. Conclusion

In this paper, we present the first comprehensive study on characterizing existing serverless workflow services, i.e., AWS Step Functions, Azure Durable Functions, Alibaba Serverless Workflow, and Google Cloud Composer. We first characterize and compare their features from six dimensions. Then, we measure the performance of these serverless workflow services under varied settings (i.e., different levels of activity complexity, data-flow complexity, and function complexity). Based on the results, some interesting findings, e.g, only under high data-flow complexity conditions will the performance of serverless workflow services has a certain impact, can be useful to guide developers and cloud providers. Finally, we report a series of findings and implications to further facilitate the development with serverless workflow services.

Acknowledgment

This work was supported by the PKU-Baidu Fund Project under the grant number 2020BD007.

References

[1] J. Carreira, P. Fonseca, A. Tumanov, A. Zhang, and R. Katz, "A case for serverless machine learning," in *Proceedings of Workshop on Systems for ML and Open Source Software at NeurIPS*, vol. 2018, 2018.

[2] E. de Lara, C. S. Gomes, S. Langridge, S. H. Mortazavi, and M. Roodi, "Hierarchical serverless computing for the mobile edge," in *Proceedings of IEEE/ACM Symposium on Edge Computing*, 2016, pp. 109–110.

[3] M. Malawski, A. Gajek, A. Zima, B. Balis, and K. Figiela, "Serverless execution of scientific workflows: experiments with HyperFlow, AWS lambda and google cloud functions," *Future Generation Computer Systems*, vol. 110, pp. 502–514, 2020.

[4] E. Jonas, Q. Pu, S. Venkataraman, I. Stoica, and B. Recht, "Occupy the cloud: distributed computing for the 99%," in *Proceedings of 2017 Symposium on Cloud Computing*, 2017, pp. 445–451.

[5] R. Chard, K. Chard, J. Alt, D. Y. Parkinson, S. Tuecke, and I. Foster, "Ripple: home automation for research data management," in *Proceedings of 37th IEEE International Conference on Distributed Computing Systems Workshops*, 2017, pp. 389–394.

[6] I. E. Akkus, R. Chen, I. Rimac, M. Stein, K. Satzke, A. Beck, P. Aditya, and V. Hilt, "Sand: towards high-performance serverless computing," in *Proceedings of the 2018 USENIX Annual Technical Conference*, 2018, pp. 923–935.

[7] G. McGrath and P. R. Brenner, "Serverless computing: design, implementation, and performance," in *Proceedings of 2017 IEEE 37th International Conference on Distributed Computing Systems Workshops*, 2017, pp. 405–410.

[8] E. Jonas, J. Schleier-Smith, V. Sreekanti, C.-C. Tsai, A. Khandelwal, Q. Pu, V. Shankar, J. Carreira, K. Krauth, N. Yadwadkar, J. E. Gonzalez, R. A. Popa, I. Stoica, and D. A. Patterson, "Cloud programming simplified: A berkeley view on serverless computing," *arXiv preprint arXiv:1902.03383*, 2019.

[9] M. Shahrad, R. Fonseca, Í. Goiri, G. Chaudhry, P. Batum, J. Cooke, E. Laureano, C. Tresness, M. Russinovich, and R. Bianchini, "Serverless in the wild: characterizing and optimizing the serverless workload at a large cloud provider," in *Proceedings of the 2020 USENIX Annual Technical Conference*, 2020, pp. 205–218.

[10] "Amazon," https://aws.amazon.com, retrieved on September 10, 2020.

[11] "Microsoft," https://azure.microsoft.com/en-us/, retrieved on September 10, 2020.

[12] "Google," https://cloud.google.com/, retrieved on September 10, 2020.

[13] "Function-as-a-service market by user type (developer-centric and operator-centric), application (web & mobile based, research & academic), service type, deployment model, organization size, industry vertical, and region - global forecast to 2021," https://www.marketsandmarkets.com/Market-Reports/function-as-a-service-market-127202409.html, retrieved on September 10, 2020.

[14] P. Datta, P. Kumar, T. Morris, M. Grace, A. Rahmati, and A. Bates, "Valve: Securing function workflows on serverless computing platforms," in *Proceedings of the 29th International Conference on World Wide Web*, 2020, pp. 939–950.

[15] "Aws step functions documentation," https://docs.aws.amazon.com/step-functions/index.html, retrieved on September 10, 2020.

[16] "Serverless workflow applicable scenarios and best practices (in chinese)," https://developer.aliyun.com/article/751573, retrieved on September 10, 2020.

[17] "What are durable functions?" https://docs.microsoft.com/en-us/azure/azure-functions/durable/durable-functions-overview?tabs=csharp, retrieved on September 10, 2020.

[18] "Aliyun serverless workflow (in chinese)," https://help.aliyun.com/product, retrieved on September 10, 2020.

[19] "Google cloud composer," https://cloud.google.com/composer?hl=en, retrieved on September 10, 2020.

[20] J. Cardoso, "Approaches to compute workflow complexity," in *Proceedings of Role of Business Processes in Service Oriented Architectures (Dagstuhl Seminar Proceedings)*. Schloss Dagstuhl-Leibniz-Zentrum für Informatik, 2006.

[21] "2018 serverless community survey: huge growth in serverless usage," https://www.serverless.com/blog/2018-serverless-community-survey-huge-growth-usage, retrieved on September 10, 2020.

[22] "Azure functions," https://azure.microsoft.com/en-us/services/functions/, retrieved on September 10, 2020.

[23] "Etl-dataprocessing using mapreduce," https://github.com/awesome-fnf/ETL-DataProcessing, retrieved on September 10, 2020.

[24] L. Wang, M. Li, Y. Zhang, T. Ristenpart, and M. Swift, "Peeking behind the curtains of serverless platforms," in *Proceedings of the 2018 USENIX Annual Technical Conference*, 2018, pp. 133–146.

[25] P. Maissen, P. Felber, P. Kropf, and V. Schiavoni, "Faasdom: A benchmark suite for serverless computing," in *Proceedings of the 14th ACM International Conference on Distributed and Event-based Systems*. ACM, 2020, pp. 73–84.

[26] P. G. López, M. Sánchez-Artigas, G. París, D. B. Pons, Á. R. Ollobarren, and D. A. Pinto, "Comparison of faas orchestration systems," in *Proceedings of 2018 IEEE/ACM International Conference on Utility and Cloud Computing Companion*, 2018, pp. 148–153.

[27] P. G. López, A. Arjona, J. Sampé, A. Slominski, and L. Villard, "Triggerflow: trigger-based orchestration of serverless workflows," in *Proceedings of the 14th ACM International Conference on Distributed and Event-based Systems*, 2020, pp. 3–14.