A Study of Orchestration Approaches for Scientific Workflows in Serverless Computing

Abdallah Elshamy*, Ahmed Alquraan*, Samer Al-Kiswany*#
*University of Waterloo, Canada
#Acronis Research, Canada
{abdallah.elshamy,ahmed.alquraan,alkiswany}@uwaterloo.ca

ABSTRACT

Scientific workflows are typically data- and compute-intensive. They consist of many stages, each of which may contain hundreds to even thousands of tasks. Traditionally, scientific workflows have been executed using the serverful computing model. Serverless computing presents an attractive alternative to the serverful computing model as it frees developers from managing and provisioning resources and offers a fine-grained pay-as-you-go pricing model. In this paper, we investigate the viability of using serverless computing to execute scientific workflows. Specifically, we discuss, implement, and evaluate three orchestration approaches for executing scientific workflows: serverful-centralized, serverless-centralized, and serverless-decentralized. This work is the first to implement and evaluate a purely serverless orchestration approach that does not require deploying a dedicated workflow manager. Our evaluation shows that serverless orchestration approaches cause a noticeable performance overhead for some workflow patterns (e.g., reduce stages) due to accessing a large amount of remote data. We propose two optimizations (i.e., prefetching file privileges and container placement) that exploit data locality to mitigate that impact. Our evaluation with the Montage application shows that a fully decentralized approach achieves a comparable performance to a serverful approach. Also, our results show that prefetching file privileges and container placement optimizations improve the performance by 26% and 44% respectively when compared to an unoptimized version.

CCS CONCEPTS

• **Software and its engineering → Cloud computing**.

KEYWORDS

serverless, scientific workflows, workflow orchestration

ACM Reference Format:
Abdallah Elshamy, Ahmed Alquraan, Samer Al-Kiswany. 2023. A Study of Orchestration Approaches for Scientific Workflows in Serverless Computing. In *The 1st Workshop on SErverless Systems, Applications and MEthodologies (SESAME '23), May 8, 2023, Rome, Italy*. ACM, New York, NY, USA, 7 pages. https://doi.org/10.1145/3592533.3592809

SESAME '23, May 8, 2023, Rome, Italy

ACM ISBN 979-8-4007-0185-6/23/05…$15.00
https://doi.org/10.1145/3592533.3592809

1 INTRODUCTION

Scientific workflows [1, 2, 3, 4, 5, 6] have seen an increase in popularity in many scientific fields (e.g., astronomy and bioinformatics) as they are being used to represent the compositions of computational tasks that form complex analysis pipelines. Complex workflows usually are composed of several stages, each of which may consist of hundreds to even thousands of computational tasks. Typically, scientific workflows are expressed as a directed acyclic graph (DAG), in which nodes correspond to tasks and edges represent data dependencies. Traditionally, scientific workflows have been executed in a serverful environment (e.g., on-premises cluster or infrastructure as a service) [7, 8, 9], in which computing and storage resources are managed and provisioned by users.

Recently, the serverless computing model has emerged as an alternative to the traditional infrastructure-as-a-service (IaaS) model. Serverless computing supports highly scalable, event-driven applications. In serverless computing, developers focus on implementing the functionality of their applications as stateless functions or containers that are triggered by events, while the cloud provider is responsible for managing and provisioning the resources. In addition, serverless adopts a pay-as-you-go billing model, in which users are only charged when their applications are invoked and run time is measured at a fine granularity (e.g., 1 ms).

Scientific workflows are typically more data- and compute-intensive than other serverless workflows. Recent characterization studies [6, 10] show that scientific workflows exhibit a significantly higher degree of parallelism compared to other serverless workflows. Furthermore, the execution time of scientific workflows is considerably longer. Lastly, the intermediate data transferred between stages is typically much larger in scientific workflows.

In this work, we focus on studying the viability of using serverless environments to execute scientific workflows. In addition, we evaluate different orchestration approaches for executing scientific workflows: serverful-centralized, serverless-centralized, and serverless-decentralized approaches (Section 3). Furthermore, our work answers the following questions: What are the challenges, advantages, and disadvantages of each approach? How the end-to-end performance of a scientific workflow is affected by the selected orchestration approach? How do other factors including file system optimizations and task-to-container mapping affect workflow performance?

To answer these questions, we perform a thorough evaluation of these orchestration approaches using the Montage [1] application, a complex astronomical workflow. Our results (Section 4) show that serverless orchestration approaches have some performance overhead for some workflow patterns, such as reduce stages. Our analysis identified that this overhead is due to accessing a

large amount of remote data. Following our analysis, we propose two locality optimizations to mitigate this performance overhead: prefetching file privileges and container placement. Prefetching file privileges is a novel optimization that overlaps the execution of a reduce stage with the parallel stage preceding it in order to acquire the privileges required to access the files needed by the reduce stage as soon as they are written. Furthermore, this optimization can be applied to both centralized and decentralized orchestration approaches. Container placement optimization targets placing tasks that share a large amount of data on the same node. Our results show that prefetching file privileges and container placement optimizations improve the performance of the unoptimized serverless version by 26% and 44%, respectively.

2 BACKGROUND AND RELATED WORK

In this section, we provide an overview of scientific workflows and highlight recent projects that focus on the execution of scientific workflows in a serverless environment.

Scientific workflows. Scientific workflows allow scientists to easily model complex scientific applications and experiments by expressing the entire data processing steps and the dependencies between these steps. Scientific workflows usually consist of multiple steps, each of which may contain a large number of independent tasks. Typically, a scientific workflow is expressed as a directed acyclic graph (DAG), in which nodes correspond to tasks and edges represent data dependencies. Different tasks typically communicate through files stored on a shared file system (e.g., CephFS [11]). A task is only executed when all its dependencies become available. Data access patterns define how the output of a task is being used by other tasks. A scientific workflow can consist of different access patterns including pipeline, broadcast, reduce, scatter, gather, and distribute patterns. Al-Kiswany et al. [12] discuss these patterns in detail.

Scientific workflows are typically more data- and compute-intensive than other serverless workflows. To demonstrate this, we contrast the characteristics of serverless workflows in a subset of Azure Durable Functions [13] production workloads [10] with the characteristics of popular scientific workflows from different fields [6]. Firstly, the number of tasks in parallel stages is much larger in scientific workflows. For instance, while the maximum number of tasks in a parallel stage in 90% of production serverless workflows is less than 5, this number is typically in the hundreds and can even reach hundreds of thousands for scientific workflows. Secondly, the execution time of scientific workflows is much longer than that of production serverless workflows. For instance, the median execution time of production serverless workflows is 5.6 sec, while the execution time of scientific workflows is typically in hours and can even reach hundreds of hours. Finally, the size of transferred intermediate data is much larger in scientific workflows. While 85% of the transferred intermediate data in production serverless workflows are of sizes $\geq$ 1 KB and the median is 8 KB, the size of transferred intermediate data in scientific workflows is typically in the hundreds of gigabytes and can even reach hundreds of terabytes.

A workflow management system (WMS) is a tool that is responsible for executing a workflow in different execution environments (e.g., a cluster, a grid, or cloud computing). WMSs are typically designed to exploit a large amount of computing and storage resources to execute the target workflow in parallel in order to reduce the total execution time. A WMS receives a workflow directed acyclic graph (DAG) as an input and then it generates an execution plan based on the workflow DAG. The generated execution plan of the tasks must adhere to the data dependencies specified by the workflow DAG. Moreover, a WMS acts as a scheduler that maps tasks to computing resources. Typically, a WMS maintains all tasks in a queue. The WMS keeps tracking of the dependencies of each task. Once the dependencies of a task are met, the WMS schedules the task on one of the available nodes. Also, a WMS is typically responsible for keep tracking of the status of all running tasks and handling failures during tasks execution. Several WMSs [14, 15, 16, 17] have been introduced to enable executing scientific workflows in various computing environments, including IaaS, FaaS, and CaaS.

Scientific workflows execution in serverless environments. Several recent efforts have looked into executing scientific workflows in serverless platforms. Malawski et al. [16] have built a WMS to execute scientific workflows using AWS Lambda and Google Cloud Functions. Their proposed approach is centralized and based on the HyperFlow engine. Similarly, SWEEP [15] is a centralized, cloud-provider agnostic WMS that supports executing a workflow using serverless functions and serverless containers. Burkat et al. [17] propose a HyperFlow-based WMS that supports AWS Lambda, AWS Fargate, and Cloud Run. The proposed WMS adopts a centralized orchestration approach with a one-to-one task to container mapping. These efforts focused only on evaluating different aspects of various serverless platforms when executing real-world scientific workflows, and do not compare the proposed solutions with other alternatives (e.g., a decentralized WMS).

These efforts focus on developing fully-fledged centralized WMSs that support various serverless providers. On the other hand, we focus on evaluating the viability of executing scientific workflows in a serverless environment. Also, we study and compare different orchestration approaches to execute scientific workflows. In addition, we discuss several optimizations that can reduce the execution time of scientific workflows in serverless environments.

Accelerating workflows in a serverless environment. Several techniques have been proposed recently to accelerate the execution of general workflows in serverless functions. Palette [18] proposes a simple technique to express locality hints to the serverless platform. Mahgoub et al. [19] propose three optimizations for workflow execution. The first optimization tries to allocate the right amount of resources for each function invocation. The second one co-locates multiple parallel instances of a function on the same virtual machine (VM). The last technique is the pre-warming of VMs with the right look-ahead time. SONIC [20] tries to optimize a workflow by dynamically selecting the best approach to pass data between different serverless functions. SONIC design considers three techniques for passing data: VM-Storage, Direct-Passing, and Remote storage. The effectiveness of these techniques was not evaluated on scientific workflows. Scientific workflows differ from general-purpose workflows; They are more complex and consist of long-running data-intensive tasks. In our work, we propose data locality optimizations that exploit distributed file systems to improve scientific workflows performance (Section 4).

3 WORKFLOW ORCHESTRATION APPROACHES

In this section, we discuss different computing models for executing scientific workflows and the orchestration approaches used in each model. First, we discuss the serverful model (e.g., IaaS) which is the traditional model used to execute scientific workflows. Then, we discuss the serverless model and illustrate two different approaches for orchestrating workflows. We compare the performance of these approaches in Section 4.

3.1 Serverful Model

Serverful Model (e.g., IaaS), represents the traditional execution model in which a set of computing nodes (i.e., workers) are responsible for executing the tasks of a workflow. In this model, the orchestration approach is centralized, in which a workflow manager is deployed on one node and responsible for orchestrating the execution of workflow tasks as shown in Figure 1. The workflow manager is aware of all computing nodes and can directly communicate with them. The workflow manager receives a workflow DAG as an input and is responsible for executing the tasks of the DAG on the worker nodes while respecting the data dependencies between tasks. Typically, a storage service is used to share data between all computing nodes, including input data, results, and temporary data. There are multiple options to realize the storage service, but the most popular one is deploying a distributed file system on the computing nodes.

The main advantage of this approach is that the workflow manager in a serverful model has full knowledge of the status of each worker node (e.g., its hardware resources and the number of assigned tasks). Hence, the workflow manager can assign tasks to nodes in an efficient manner to improve resource utilization and reduce end-to-end execution time. On the other hand, in this approach, developers are responsible for the management and provisioning of the computing and storage resources, which complicates auto-scaling resources with workflows with dynamic resources demand.

3.2 Serverless Model

The serverless model differs from the serverful model in many ways. First, computing resources are not fixed as the serverless platform can dynamically scale up and down computing resources based on demand. Second, a workflow manager is not aware of the location of each running task. Finally, mapping tasks to workers is more complex. The simplest approach is to assign one task per container. However, this approach can consume more resources and take more time in parallel stages. If the tasks of the parallel stages are short, then creating a new container for each task can take more time than reusing containers for multiple requests. Hence, executing data-intensive, scientific workflows in a serverless platform efficiently is more challenging.

In order to execute a workflow in a serverless environment, each task has to be implemented as a serverless execution unit (i.e., a function in the case of FaaS and a container in the case of CaaS). In this work, we focus on serverless containers as they offer more control over the execution environment making them more suitable for scientific applications. Typically, a storage service is employed to share data between different containers. This storage service can be implemented as a distributed file system or any other cloud storage (e.g., Amazon S3 [21]). A container reads input data, executes the task, and then writes output data to the storage service.

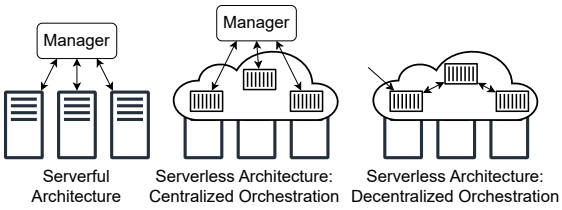

Figure 1: Different orchestration approaches for scientific workflows

In the following, we discuss two approaches for orchestrating tasks of a workflow in serverless environments:

Centralized orchestration approach. As shown in Figure 1, the workflow tasks are orchestrated by a centralized workflow manager. The workflow manager is responsible for sending requests to execute the workflow tasks, checking the responses of these requests, and triggering subsequent stages based on those responses while respecting the data dependencies between different tasks. The main issue with this approach is that the workflow manager has to be deployed on dedicated resources as it is a stateful, long-running process that has to be running till the workflow execution is over. Hence, the workflow manager cannot be deployed as a serverless container.

Decentralized orchestration approach. In this approach, the logic of the workflow manager is embedded into the workflow stages. This makes every stage responsible for triggering the next stage as shown in Figure 1. Hence, this approach is considered a purely serverless approach as it does not require the deployment of a workflow manager on a dedicated computing node. However, it requires considerable development effort in order to implement the functionality of the workflow manager inside containers. Also, this approach complicates handling some workflow patterns. For example, implementing reduce stages is challenging in this approach. A reduce stage should start after all parallel tasks in the previous stage finish. However, it is hard to know when the last parallel task finishes without using a dedicated workflow manager. Moreover, the only way to infer that parallel tasks are done is by checking their output files. If all expected output files exist, then all parallel tasks are done. Otherwise, some tasks are still running.

We explore two approaches to implement the reduce stage. The first one uses a special container that is responsible for triggering the reduce stage. This container is triggered when the first parallel task completes. When the container is triggered, it checks if the output files of all tasks of the parallel stage exist. If all files exist, it triggers the reduce stage. Else, it sends a request to itself to keep spinning. Although this approach is simple, it has two main problems. First, due to performance variation in parallel tasks, this container might get triggered early, and if other parallel tasks take a long time, the container will keep performing file system operations to check the output files of the parallel tasks, which might stress the distributed file system and cause its performance to deteriorate affecting the execution time of the whole workflow. Second, if one

of the parallel tasks fails permanently, the container will not be aware of this failure and it will keep spinning forever.

The second approach to implement the reduce stage is to modify the parallel tasks in the stage before the reduce stage such that every parallel task checks if all output files exist. If so, the task will trigger the reduce stage. However, in order to guarantee that the reduce stage will be triggered only once, the task uses an atomic file system operation (e.g., renaming a file), and that task will trigger the reduce stage. The main advantages of this approach are that the number of trials to trigger the reduce stage is limited by the number of parallel tasks, the number of file system operations is bounded, and the reduce stage is not affected by failures of parallel tasks. However, this approach requires the use of a distributed file system that supports atomic operations. We use this approach in our evaluation as the number of checks that we make is limited by the number of parallel tasks which gives a more deterministic performance.

4 EVALUATION

We evaluate different orchestration approaches by comparing the end-to-end and per-stage execution time of Montage, a real-world scientific workflow (Section 4.1). We identify optimization opportunities and evaluate their benefits in Section 4.2. In Section 4.3, we show how the decision of assigning tasks to containers can significantly impact performance. Finally, we show the effect of cold starts in Section 4.4.

Testbed. We conducted our experiments using 11 CloudLab [22] nodes at the Wisconsin data center. Each node has two Intel Xeon Silver 10-core CPUs and 196 GB of RAM. We use Knative [23] to implement a serverless environment. Knative is a platform-agnostic solution for running serverless containers in a Kubernetes environment. We deploy Knative v1.8.0 on a Kubernetes environment v1.24.6. We do not limit the CPU and RAM a container can use in our experiments.

Distributed File system. We use a distributed file system as a storage service to share files between worker nodes. We choose to use a distributed file system for two reasons. First, it is the most popular method used in HPC applications. Second, it is the simplest method to integrate with all different approaches, which ensures that any performance difference is mainly due to the used approach. The file system we use is CephFS [11], the `Pacific` release (version 16.2.1). CephFS is a highly available and performant POSIX-compliant file system. We use CephFS as it is the de facto choice for HPC scratch spaces and distributed workflows shared storage. We configure CephFS to use RAM disks to ensure that the results we obtain are not affected by the performance of persistent disks. Each worker node has a 50 GB RAM disk.

Workflow. We use the Montage application [1] in our experiments. Montage is an open-source astronomy workflow. The structure of this workflow is shown in Figure 2. We can see that Montage is a complex workflow with different types of stages, such as sequential stages, parallel stages, and reduce stages. We use the `"2MASS J"` dataset with the following parameters: size 4 and location `"M 101"`. This creates a workflow with 4034 tasks, which include 713 `mProjectPP` parallel tasks, 2602 `mDiffFit` parallel tasks, and 713 `mBackground` parallel tasks. We report the average of 15 trials for each experiment. The standard deviation of the end-to-end execution time is less than 9% for most of the experiments and 15% for the cold start experiments.

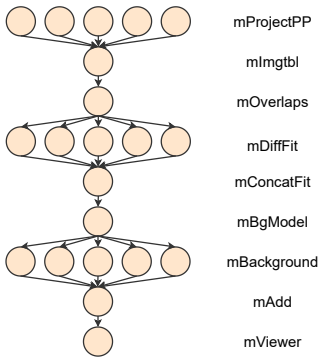

Figure 2: The structure of Montage workflow.

Alternatives. We compare the following orchestration approaches:

- Sequential: This alternative implements the workflow as sequential steps without any parallelism. In this alternative, the workflow is executed on one node. Also, it does not use a distributed file system rather it only uses the local RAM disk. This alternative represents a baseline for the other alternatives.
- Serverful-Centralized: This alternative represents the execution of the workflow in a serverful environment using a centralized workflow manager. We use PyFlow [12] to implement the workflow manager. PyFlow is a workflow management system that supports the parallel execution of workflow tasks on worker nodes while respecting the dependencies between different tasks as specified by the workflow DAG. PyFlow offers a simple representation of how workflow DAGs are currently executed in a serverful environment. PyFlow runs on one node and utilizes other nodes to execute the tasks.
- Serverless-Centralized: This alternative uses a centralized workflow manager to orchestrate the execution of workflow tasks in the serverless environment. One node runs the workflow manager, while the rest of the nodes are used to run containers that execute the tasks. We implemented a simple workflow manager using Python.
- Serverless-Decentralized: In this approach, we modify the workflow tasks to integrate the functionality of the workflow manager within the tasks themselves. That is, a task will trigger the next tasks in a workflow. Only 10 worker nodes are used in this alternative to ensure that all alternatives use the same amount of computing resources.

4.1 Comparing Different Orchestration Approaches

We start by comparing the performance of the Montage workflow using different orchestration approaches. Figure 3 shows the average execution time of each stage and the end-to-end execution time of different alternatives. Results show that parallel alternatives

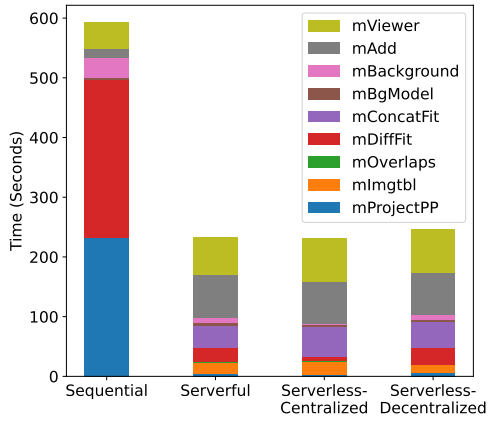

Figure 3: Average execution time of each stage using different orchestration approaches.

(i.e., serverful-centralized, serverless-centralized, and serverless-decentralized) improve the performance by at least 58% compared to the sequential alternative. Also, results show that serverless-centralized and serverless-decentralized alternatives achieve a comparable performance to the serverful-centralized alternative. In the following, we detail these results and discuss our observations.

Overhead of the orchestration approach. We notice that in the serverless-centralized alternative, the execution time of parallel stages is lower than that of the serverful-centralized alternative. For instance, the execution time of the `mPorjectPP` stage using the serverless-centralized and the serverful-centralized alternatives is 3.48 seconds and 4.89 seconds, respectively. This performance difference is justified by the difference in the behaviour of the workflow manager in these alternatives. In the serverful-centralized alternative, the workflow manager is responsible for monitoring the worker nodes to keep track of the number of tasks that each worker node is handling in order to efficiently distribute the tasks among worker nodes, which causes some performance overhead for parallel stages. While in the serverless-centralized alternative, the workflow manager is only responsible for communicating with the serverless platform by sending requests to execute tasks and receiving responses from these tasks.

We notice a significant slowdown in the parallel stages for the serverless-decentralized alternative compared to the serverless-centralized alternative. For instance, for `mProjectPP`, `mDiffFit`, and `mBackground` stages, the serverless-decentralized alternative execution time is higher by 1.9, 3.9, and 5.7 times, respectively, when compared to the serverless-centralized alternative. This slowdown is due to the overhead resulting from integrating the workflow manager within the functionality of the parallel tasks. Each parallel task, after finishing its original functionality, communicates with CephFS to check the existence of the output files of all parallel tasks. This mechanism results in flooding CephFS with many requests which causes a noticeable performance overhead. We note that this overhead increases the end-to-end execution time by only 6.48% as the workflow is dominated by long sequential stages. However, for workflows that are dominated by parallel stages, the impact on the end-to-end execution time is expected to be more noticeable.

Effect of distributed file systems. Figure 3 shows that serverful and serverless alternatives suffer from large execution time in the reduce stages and stages that process large files when compared to the sequential alternative. For instance, for `mAdd`, and `mImgtbl` stages, the serverless-decentralized alternative execution time is higher by 4.7 and 15.5 times, respectively, when compared to the sequential alternative. This performance degradation is due to optimizations that are typically implemented in distributed file systems; CephFS utilizes file system capabilities (i.e., privileges) to optimize write and read operations. If a file is being written by only one node, CephFS grants write and buffer capabilities exclusively to this writer node. These capabilities allow the writer node to write data to its local buffer, which is cached and used to serve reads coming from the same node. However, if another node tries to access (e.g., read or write) the same file, CephFS revokes the granted capabilities from the writer node to force it to flush its buffer to storage nodes. After that, CephFS grants capabilities to the other node to allow it to access the file. CephFS uses this approach to guarantee data consistency between different nodes. Stages that read many input files (e.g., reduce stages) have to wait till the data written by all parallel tasks of the previous stage is flushed to storage nodes. Also, stages that read large files (e.g., `mViewer`) suffer from the same issue as large files are typically partitioned into chunks that are stored on different storage nodes.

4.2 Locality Optimizations

In this section, we present and evaluate two optimizations that can mitigate the effects of using a distributed file system in both serverless-centralized and serverless-decentralized alternatives:

Prefetching file privileges. This optimization prefetches file privileges at the reduce task. To achieve this, we added a new task that only opens a given file. Opening a file forces the previous node that holds the privileges of the file to flush its buffer and for the new node to acquire file privileges. We schedule the new task on the container that will run the reduce task. This approach forces the nodes that run the parallel tasks to flush their buffers and results in acquiring the needed privileges for the output files at the reducer. Note that this optimization overlaps the process of acquiring file privileges by the reducer with parallel tasks that are still running.

Container placement. In this optimization, containers of tasks that share a large amount of data are placed on the same node (e.g., a parallel stage and the following reduce stage). This optimization improves the performance significantly as it reduces the amount of data that is accessed remotely by tasks; Tasks write output data to local buffers and subsequent tasks read input data from these buffers. Although this optimization improves performance significantly (Figure 4), there are multiple problems with using it to optimize reduce patterns. First, placing all parallel tasks of a stage on the same node may reduce resource utilization and increase the execution time of that stage as all tasks are competing over the same computing resources. Second, this optimization has to be implemented by the serverless platform as applications do not have direct access to the underlying infrastructure.

We study the effect of each of these optimizations on performance. Figure 4 shows the end-to-end execution time and the execution time of each stage of the Montage workflow using the two optimizations. Results show that these optimizations reduce the

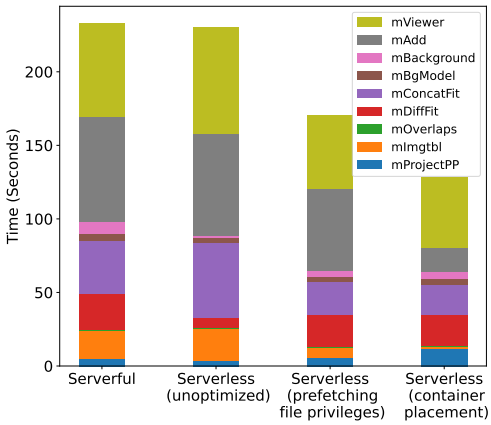

Figure 4: The effect of locality optimizations on the workflow execution time

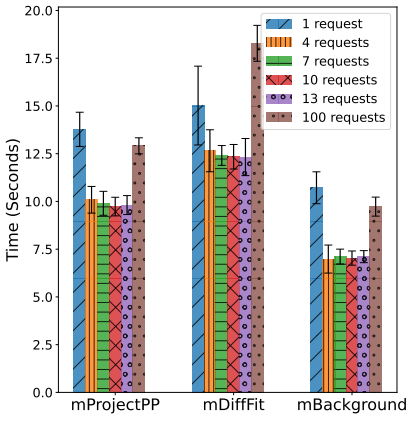

Figure 5: The effect of the number of concurrent requests on the execution time of parallel stages.

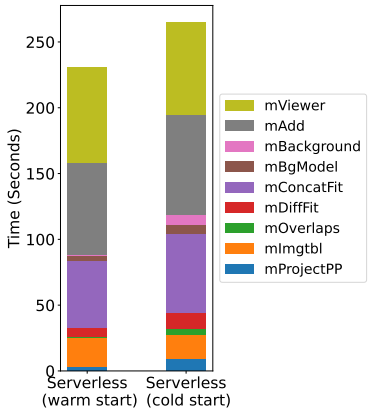

Figure 6: The effect of containers cold starts on the workflow execution time.

end-to-end execution time. Specifically, compared to the unoptimized serverless alternative, prefetching file privileges and container placement optimizations reduce the end-to-end execution time of the unoptimized version by 26% and 44%, respectively. The optimizations also reduce the variation in the end-to-end execution time with a standard deviation of less than 2%. Furthermore, container placement optimization reduces the execution time of reduce stages significantly when compared to the unoptimized version (e.g., `mImgtbl` by 95.6%, `mConcatFit` by 59.95%, and `mAdd` by 76.88%). However, it increases the execution time of parallel stages `mProjectPP`, `mDiffFit`, and `mBackground` by 2.5, 1.92, and 3.15 times, respectively. This performance reduction is due to resource contention as all parallel tasks are executed on one node.

4.3 Task to Container Mapping

In this experiment, we study the effect of task-to-container mapping on the performance of the parallel stages. Knative supports configuring the number of concurrent requests a container can handle. If all running containers have reached their concurrency limit, Knative spins a new container to serve any future requests. Figure 5 shows the execution time of each parallel stage for different numbers of concurrent requests. Results show that a simple one-to-one task to container mapping (i.e., one request for each container which results in spinning up 143 containers in Figure 5) increases the execution time of the parallel stages `mProjectPP`, `mDiffFit`, and `mBackground` by 40.4%, 21.89%, and 50.75%, respectively when compared to the best configuration (i.e., 13 concurrent requests which results in spinning up 11 containers). This performance deterioration is due to spinning a large number of cold containers as we hit a cold start with every new container created. Also, results show that increasing the container concurrency to a large value (100 concurrent requests which results in spinning up two containers) results in provisioning few containers and does not utilize the elasticity offered by the serverless platform resulting in a longer execution time for parallel stages. Since parallel tasks in the Montage workflow are short, allowing a container to handle multiple concurrent tasks takes less time than creating new containers. Hence, results suggest that there is a sweet configuration spot that results in lower execution time and better resource utilization. We did a similar experiment for determining the best number of warm containers. In all other experiments, we use the best configurations we acquired empirically from those experiments which are seven warm containers for each parallel stage in the case of a warm start and 13 concurrent requests per container in the case of a cold start.

4.4 Effect of Cold Starts

This section evaluates the effect of cold starts of containers on the performance. Cold starts is a known issue and its impact on serverless functions has been evaluated by previous efforts [24]. However, the tasks of scientific workflows differ from general-purpose serverless functions as they are long-running tasks that consume and produce a large amount of data. Figure 6 shows the average end-to-end execution time and the execution time of each stage of the Montage workflow using the cold start and warm start versions. We use the best configuration for both versions. For the warm start version, we configure the number of possible concurrent requests to a high value in order to prevent the creation of any new containers and avoid any cold starts. For the cold start version, we store the containers' images on the nodes. However, the cache and the swap area are cleared before each experiment to ensure that we always hit a cold start. Figure 6 shows that the cold starts increased the total end-to-end execution time by 14.89%. We notice that the effect of cold starts on some of the sequential and reduce stages is non-noticeable (e.g., `mAdd` and `mViewer`). This is because these stages run for a long time which diminishes the effect of cold starts. However, for the parallel stages, we notice a significant slowdown (e.g., 2.82 times higher for the `mProjectPP` stage). This is due to cascaded cold starts as we hit a cold start with every new container being created.

5 DISCUSSION

Viability of using serverless computing to execute scientific workflows. Our evaluation shows that serverless computing is a viable approach for executing scientific workflows as it offers comparable performance to the traditional serverful approach. We plan to extend our study with similar evaluations for resource utilization

and cost. Also, the design of the proposed locality optimizations suggests that it is critical to coordinate the scheduling decisions and the file system optimizations.

Impact of the optimizations. The optimizations introduced in this study can have a negative impact on the end-to-end execution time or resource utilization under some workflows. The "container placement" optimization may result in increasing the end-to-end execution time if the increase in the time of the parallel stages is higher than the time reduction for reduce stages. Furthermore, if the majority of running containers are of a few parallel stages, these containers will be placed on a few nodes resulting in low resource utilization. For the "prefetching file privileges " optimization, we might face some issues if the duration of the parallel tasks has a large variance. In this case, the reducer might be provisioned early, wait for a while without anything to do as the other parallel tasks are still running, then exits without executing the reducer task. For this scenario, it might be better to just open all the files in parallel before the parallel stage finishes.

Implementing the decentralized orchestration approach. Due to its distributed nature, implementing, modifying, and debugging the decentralized orchestration approach requires a considerable amount of effort. This is calling for the need for further tools and libraries that offer standard components for implementing common patterns in workflows while transparently handling common issues such as logging and failures. Regarding the implementation of parallel tasks, the used method to check the existence of all output files of parallel tasks can significantly affect the execution time. In our implementation, we check only the number of the output files. However, if a more sophisticated method is used (e.g., checking the name of the files), the performance deterioration will be much worse as it will generate a higher load on the distributed file system.

6 CONCLUSION AND FUTURE WORK

Our study investigates the viability of using serverless computing in executing scientific workflows. We implement and evaluate three different orchestration approaches using the Montage application. The results show that serverless computing offers a comparable performance compared to the serverful approach. The results also show that all approaches suffer from performance overhead in certain stages. We investigate the causes of that overhead, and we then propose two optimizations (i.e., prefetching file privileges and container placement) to mitigate that overhead. For future work, we intend to extend our study with other workflows that have different patterns and create a framework that leverages the insights we find in our study.

REFERENCES

[1] Daniel Katz, G. Berriman, John Good, Anastasia Laity, Ewa Deelman, Carl Kesselman, Gurmeet Singh, Mei-Hui Su, Thomas Prince, and Roy Williams. Montage: A grid portal and software toolkit for science-grade astronomical image mosaicking. *International Journal of Computational Science and Engineering*, 4, 05 2010.

[2] Suyash Shringarpure, Andrew Carroll, Francisco De La Vega, and Carlos Bustamante. Inexpensive and highly reproducible cloud-based variant calling of 2,535 human genomes. *PloS one*, 10:e0129277, 06 2015.

[3] Oleg Trott and Arthur J. Olson. Autodock vina: Improving the speed and accuracy of docking with a new scoring function, efficient optimization, and multithreading. *Journal of Computational Chemistry*, Jun 2009.

[4] William L. Poehlman, Mats Rynge, D. Balamurugan, Nicholas Mills, and Frank A. Feltus. Osg-kinc: High-throughput gene co-expression network construction using the open science grid. In *2017 IEEE International Conference on Bioinformatics and Biomedicine (BIBM)*, pages 1827–1831, 2017.

[5] Yang Liu, Saad Khan, Juexin Wang, Mats Rynge, Yuanxun Zhang, Shuai Zeng, Shiyuan Chen, Joao Vitor Maldonado dos Santos, Babu Valliyodan, Prasad Calyam, Nirav Merchant, Henry Nguyen, Dong Xu, and Trupti Joshi. Pgen: Large-scale genomic variations analysis workflow and browser in soykb. *BMC Bioinformatics*, 17:337, 10 2016.

[6] Gideon Juve, Ann Chervenak, Ewa Deelman, Shishir Bharathi, Gaurang Mehta, and Karan Vahi. Characterizing and profiling scientific workflows. *Future Generation Computer Systems*, 29(3):682–692, 2013. Special Section: Recent Developments in High Performance Computing and Security.

[7] Jens-Sönke Vöckler, Gideon Juve, Ewa Deelman, Mats Rynge, and Bruce Berriman. Experiences using cloud computing for a scientific workflow application. In *Proceedings of the 2nd International Workshop on Scientific Cloud Computing*, ScienceCloud '11, page 15–24, New York, NY, USA, 2011. Association for Computing Machinery.

[8] Gideon Juve, Ewa Deelman, Karan Vahi, Gaurang Mehta, Bruce Berriman, Benjamin P. Berman, and Phil Maechling. Scientific workflow applications on amazon ec2. In *2009 5th IEEE International Conference on E-Science Workshops*, pages 59–66, 2009.

[9] Janez Kranjc, Vid Podpečan, and Nada Lavrač. Clowdflows: a cloud based scientific workflow platform. In *Machine Learning and Knowledge Discovery in Databases: European Conference, ECML PKDD 2012, Bristol, UK, September 24-28, 2012. Proceedings, Part II 23*, pages 816–819. Springer, 2012.

[10] Ashraf Mahgoub, Edgardo Barsallo Yi, Karthick Shankar, Eshaan Minocha, Sameh Elnikety, Saurabh Bagchi, and Somali Chaterji. Wisefuse: Workload characterization and dag transformation for serverless workflows. In *Abstract Proceedings of the 2022 ACM SIGMETRICS/IFIP PERFORMANCE Joint International Conference on Measurement and Modeling of Computer Systems*, SIGMETRICS/PERFORMANCE '22, page 57–58, New York, NY, USA, 2022. Association for Computing Machinery.

[11] Ceph file system. https://docs.ceph.com/en/pacific/cephfs/index.html, 2023.

[12] Samer Al-Kiswany, Lauro B. Costa, Hao Yang, Emalayan Vairavanathan, and Matei Ripeanu. A cross-layer optimized storage system for workflow applications. *Future Generation Computer Systems*, 75:423–437, 2017.

[13] Microsoft. Durable functions overview - azure | microsoft learn. https://learn.microsoft.com/en-us/azure/azure-functions/durable/durable-functions-overview?tabs=csharp-inproc, 2023.

[14] Ewa Deelman, Karan Vahi, Gideon Juve, Mats Rynge, Scott Callaghan, Philip J. Maechling, Rajiv Mayani, Weiwei Chen, Rafael Ferreira da Silva, Miron Livny, and Kent Wenger. Pegasus, a workflow management system for science automation. *Future Generation Computer Systems*, 46:17–35, 2015.

[15] Aji John, Kristiina Ausmees, Kathleen Muenzen, Catherine Kuhn, and Amanda Tan. Sweep: Accelerating scientific research through scalable serverless workflows. In *Proceedings of the 12th IEEE/ACM International Conference on Utility and Cloud Computing Companion*, UCC '19 Companion, page 43–50, New York, NY, USA, 2019. Association for Computing Machinery.

[16] Maciej Malawski, Adam Gajek, Adam Zima, Bartosz Balis, and Kamil Figiela. Serverless execution of scientific workflows: Experiments with hyperflow, aws lambda and google cloud functions. *Future Generation Computer Systems*, 110:502–514, 2020.

[17] Krzysztof Burkat, Maciej Pawlik, Bartosz Balis, Maciej Malawski, Karan Vahi, Mats Rynge, Rafael Ferreira da Silva, and Ewa Deelman. Serverless containers – rising viable approach to scientific workflows. In *2021 IEEE 17th International Conference on eScience (eScience)*, pages 40–49, 2021.

[18] Mania Abdi, Sam Ginzburg, Charles Lin, Jose M Faleiro, Íñigo Goiri, Gohar Irfan Chaudhry, Ricardo Bianchini, Daniel S. Berger, and Rodrigo Fonseca. Palette load balancing: Locality hints for serverless functions. In *Proceedings of the 18th European Conference on Computer Systems (EuroSys)*. ACM, May 2023.

[19] Ashraf Mahgoub, Edgardo Barsallo Yi, Karthick Shankar, Sameh Elnikety, Somali Chaterji, and Saurabh Bagchi. ORION and the three rights: Sizing, bundling, and prewarming for serverless DAGs. In *16th USENIX Symposium on Operating Systems Design and Implementation (OSDI 22)*, pages 303–320, Carlsbad, CA, July 2022. USENIX Association.

[20] Ashraf Mahgoub, Karthick Shankar, Subrata Mitra, Ana Klimovic, Somali Chaterji, and Saurabh Bagchi. SONIC: Application-aware data passing for chained serverless applications. In *2021 USENIX Annual Technical Conference (USENIX ATC 21)*, pages 285–301. USENIX Association, July 2021.

[21] AWS. Amazon s3 - cloud object storage. https://aws.amazon.com/s3/, 2023.

[22] Dmitry Duplyakin, Robert Ricci, Aleksander Maricq, Gary Wong, Jonathon Duerig, Eric Eide, Leigh Stoller, Mike Hibler, David Johnson, Kirk Webb, et al. The design and operation of cloudlab. In *USENIX Annual Technical Conference*, pages 1–14, 2019.

[23] Knative. Knative: Serverless containers in kubernetes environments. https://knative.dev, 2023.

[24] Johannes Manner, Martin Endreß, Tobias Heckel, and Guido Wirtz. Cold start influencing factors in function as a service. In *2018 IEEE/ACM International Conference on Utility and Cloud Computing Companion (UCC Companion)*, pages 181–188, 2018.