SWEEP: Accelerating Scientific Research Through Scalable Serverless Workflows

Aji John*
University of Washington
Seattle, Washington
ajijohn@uw.edu

Kristiina Ausmees*
Uppsala University
Uppsala, Sweden
kristiina.ausmees@it.uu.se

Kathleen Muenzen
University of Washington
Seattle, Washington
kmuenzen@uw.edu

Catherine Kuhn
University of Washington
Seattle, Washington
ckuhn@uw.edu

Amanda Tan
University of Washington
Seattle, Washington
amandach@uw.edu

ABSTRACT

Scientific and commercial applications are increasingly being executed in the cloud, but the difficulties associated with cluster management render on-demand resources inaccessible or inefficient to many users. Recently, the serverless execution model, in which the provisioning of resources is abstracted from the user, has gained prominence as an alternative to traditional cyberinfrastructure solutions. With its inherent elasticity, the serverless paradigm constitutes a promising computational model for scientific workflows, allowing domain specialists to develop and deploy workflows that are subject to varying workloads and intermittent usage without the overhead of infrastructure maintenance. We present the Serverless Workflow Enablement and Execution Platform (SWEEP), a cloud-agnostic workflow management system with a purely serverless execution model that allows users to define, run and monitor generic cloud-native workflows. We demonstrate the use of SWEEP on workflows from two disparate scientific domains and present an evaluation of performance and scaling.

CCS CONCEPTS

• **Information systems** → **Computing platforms**; • **Computer systems organization** → **Cloud computing**; • **Applied computing** → Environmental sciences; Computational genomics.

KEYWORDS

scalable workflows, serverless computing, workflow management systems, variant calling, satellite image analysis

ACM Reference Format:

Aji John, Kristiina Ausmees, Kathleen Muenzen, Catherine Kuhn, and Amanda Tan. 2019. SWEEP: Accelerating Scientific Research Through Scalable Serverless Workflows. In *IEEE/ACM 12th International Conference on Utility and Cloud Computing Companion (UCC '19 Companion)*, December 2–5, 2019,

*Both authors contributed equally to this research.

Permission to make digital or hard copies of part or all of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for third-party components of this work must be honored. For all other uses, contact the owner/author(s).

UCC '19 Companion, December 2–5, 2019, Auckland, New Zealand

© 2019 Copyright held by the owner/author(s).

ACM ISBN 978-1-4503-7044-8/19/12.

<https://doi.org/10.1145/3368235.3368839>

Auckland, New Zealand. ACM, New York, NY, USA, 8 pages. <https://doi.org/10.1145/3368235.3368839>

1 INTRODUCTION

Serverless computing has recently gained popularity as an alternative to the traditional Infrastructure as a Service (IaaS) model for cloud-based computation [1, 2]. With billing based on consumption and inherent elasticity of resources, the serverless paradigm relieves the user from provisioning and maintaining the computing infrastructure, and many commercial cloud vendors, including Amazon Web Services (AWS), Google Cloud Platform (GCP), Microsoft Azure, and IBM now offer serverless options. Function as a Service (FaaS) platforms like AWS Lambda, Google Cloud Functions and Azure Functions allow code to be executed as "stateless" functions, meaning the user is unaware of the execution environment and there is no persistent maintenance of state between invocations. Some providers also offer serverless options to run containers in a stateless manner on Container as a Service (CaaS) platforms, e.g. AWS Fargate and Azure Container Instances.

The workflow paradigm constitutes a common model for describing sets of interdependent steps involved in various academic and industrial applications. A variety of workflow management systems (WMS) are available, addressing different aspects of defining, executing, monitoring and evaluating workflows. The rise in popularity of cloud computing has also led to an increased use in workflow-based applications, but many WMS that are commonly used within the scientific domain still rely on the user to set up and configure clusters or IaaS cloud resources.

Our proposed work, the Serverless Workflow Enablement and Execution Platform (SWEEP), is a WMS built entirely on a serverless backbone that provides an end-to-end solution for defining, orchestrating, and running workflows without the need to manage compute resources. SWEEP is intended to be cloud-provider agnostic and uses a workflow definition formalism that is adapted to the paradigm of stateless tasks deployed in serverless infrastructures, allowing users to define generic cloud-native workflows that are inherently scalable. In this study, we demonstrate the utility of SWEEP by executing and evaluating workflows from two different scientific domains: a canonical genomics workflow for germline variant calling and a geosciences workflow that uses fine-resolution satellite imagery to analyze surface reflectance in lakes.

2 BACKGROUND AND RELATED WORK

In many scientific fields, the increasingly data-driven nature of research has seen a rise in popularity of workflows for representing the compositions of computational tasks that form complex analysis pipelines, and has led to the development of a wide variety of systems for managing workflows [3]. Textual workflow languages are low-level tools intended for users that are comfortable with scripting and programming, and range from lightweight solutions like rule-based shell scripts, e.g. Snakemake [4] and functional scripting languages like Swift [5], to flexible frameworks such as Pegasus [6] that allow for abstract workflow definition and execution using heterogeneous compute resources. Graphical WMS like Kepler and Apache Taverna [7, 8], on the other hand, include a graphical user interface (GUI) that helps streamline the workflow creation process and allows domain scientists to define, monitor and analyze complex workflows with minimal effort.

WMS can also be divided into general-purpose systems that allow for execution of arbitrary computational tasks, and domain-specific systems geared towards a particular research field. In the field of bioinformatics, Galaxy [9] is a widely-used WMS that provides a web-based interface, pre-built components for common tasks, and public data sharing features.

Another means of characterizing WMS is by the type of computational infrastructures they support. The increasing use of cloud computing has placed demands on WMS to function within the cloud framework. Distributed computing frameworks like Spark [10] and Hadoop YARN [11] have been used to develop workflow solutions such as Hi-WAY [12] and SAASFEE [13], and most WMS that were initially developed for cluster and grid computing architectures now also support execution in cloud environments. Because many scientific workflows involve tasks that rely on specialized third-party software, workflow execution using heterogeneous infrastructures is nontrivial. Automation tools such as Terraform [14] can be used to configure computing infrastructure that is dependent on third-party software, but frequent updates and version incompatibilities between tools make deployment and reproducibility of workflows challenging [15].

To address this issue, WMS that encapsulate packages and code using container technology, e.g. Docker [16], have emerged. Such platforms include Nextflow [17], Bwb [18] and Pachyderm [19]. While containerization addresses many of the issues outlined above and has facilitated the execution of generic workflows in a language- and cloud-agnostic manner [20, 21], IaaS services still require users to deploy and manage clusters. Resource management tools like Docker Swarm and Kubernetes are mature and widely used technologies that help manage container orchestration and even support auto-scaling of resources, but they still require an installation and configuration process that may be cumbersome, or in the case of managed solutions, expensive.

Cost and performance analyses have shown that serverless architectures can provide a realistic alternative to these traditional solutions in various scenarios. Performance benchmarking of a microservices-based application showed that infrastructure costs can be reduced by using AWS Lambda compared to monolithic architectures or user-implemented cloud-based solutions [22]. A

groundbreaking use of serverless computing is embodied by PyWren [23], which is built entirely on AWS Lambda functions. Benchmarking of the execution model showed that write bandwidth to AWS S3 can compete with that of local SSD on EC2 instances, and that linear scaling of FLOPs and I/O throughput of up to 3000 workers can be reached. A linear-algebra extension NumPyWren [24] has also been developed on top of PyWren.

While such frameworks have laid the foundation for serverless computing in scientific applications, its use for workflows comprising heterogeneous black-box tasks remains under-explored. Many cloud providers offer some form of workflow orchestration tool, e.g. Azure Logic Apps and AWS Step Functions, but these are vendor-specific and often come at a premium. For example, an AWS Lambda invocation costs \$0.0000002 per request, while a state transition in Step Functions is 125 times more expensive at \$0.000025. Open-source alternatives include Fission Workflows [25], which run on the Fission FaaS framework for Kubernetes, DEWE v3 [26], and a prototype extension of Hyperflow described in [27], where the latter two represent hybrid approaches of FaaS and dedicated resources. The contribution of SWEEP is thus a generic WMS built completely on serverless platforms that is not bound to a single provider and supports functions as well as containers as executable units, without requiring user-managed compute infrastructure.

3 DESIGN AND IMPLEMENTATION

At the time of writing, SWEEP is in beta version and available upon request as a cloud server and associated API ¹. Currently supported compute engines are AWS Lambda and Fargate, with extension to other providers under development. More information about the API, along with all task and workflow definitions from the case studies in this paper, is available at the SWEEP documentation page ². In the remainder of this section, we briefly outline the main features of SWEEP and provide a high-level description of options for creating workflows. For full descriptions of workflow and task definitions, we refer to the documentation.

3.1 Workflow Definition

SWEEP workflows are abstractly represented using the Directed Acyclic Graph (DAG) formalism, in which tasks correspond to nodes, and an edge from node X to node Y signifies that X must complete before Y can be executed. Users define workflows by means of a data serialization language which is parsed into the graph representation. In order to capture the unique requirements for defining and executing serverless workflows comprised of stateless tasks, a new definition and execution model has been developed within SWEEP.

The goal was to design a system that hands all control of execution runtimes over to the cloud provider, makes no assumptions about data persistence and maintains a black-box view of tasks, while following common design principles of other workflow definition languages in order to retain portability and familiarity for users of e.g. Common Workflow Language (CWL) and Workflow Description Language (WDL). Thus, SWEEP supports constructs such as scatter and gather, conditionals, and the use of expressions

¹<https://sweep.run>

²<https://docs.sweep.run>

for control of data flow. However, SWEEP also implements more complex flow control such as nested scatter and multi-level gather, and unlike CWL, in which expressions are written in JavaScript, SWEEP is purely Python-based.

3.2 Tasks

Tasks comprise the executable units of a workflow, or the nodes in the DAG representation. In SWEEP, there are two types of tasks, function-based and container-based, which correspond directly to the serverless execution models FaaS and CaaS. Users define tasks by packaging code, dependencies and/or runtime information as required by the cloud provider (for example, in a zip-file or a Docker image) and registering the package with SWEEP. As previously stated, SWEEP workflows follow the stateless paradigm in which tasks have no affinity to the underlying compute infrastructure and therefore no persistent state other than what is explicitly externalized, using e.g. object stores or databases. The WMS does not impose directives for where data should be stored or make any other assumptions about task functionality.

3.3 Execution Control

SWEEP offers several workflow constructs for static and dynamic control of execution. Dynamic parallelism is implemented using the task properties *scatter* and *follow*, described below. By combining these, custom multi-level scatter and gather behavior can be defined. Figure 1 shows an example of a nested scatter followed by three different types of gather.

The *scatter* property causes a task Y to be multiplied according to a list-type output of a predecessor task X , so that each new copy Y_i of the task Y receives one item of the list as input. The behavior of descendants Z of the scattered task Y is controlled by the *follow* property. If a node Z is specified to follow an ancestor node U , Z will be multiplied to the same value that U was multiplied by, and for every U_i and every descendant of U_i for which there was a path from U to Z in the original workflow definition, there will be a path from U_i to Z_i . Intuitively, Z_i gathers all of the aggregated output from U_i . The default behavior (if no follow value is specified), is that one copy of the task Z is spawned, and there is an edge to Z from every node that is a predecessor of Z in the original workflow definition.

Other control structures include specifying conditions that must be fulfilled for tasks to execute, as well as delay, timeout and concurrency constraints. Error handling behavior can be controlled in a task-specific fashion by limiting the number of retries, and specifying whether or not a task failure should be ignored or should halt workflow execution.

3.4 Flow of Information

On the workflow level, information passes along the edges of the DAG, and every task has access to the data that was generated by each of its predecessors. On the execution level, the information that can be propagated to the task runtime depends on the interface supported by the task type. Functions are invoked with arguments and can return values. Containers do not support input or output, but it is possible to set their environment variables and modify the command that is run at entry. In addition to task-specific behavior,

Figure 1: Multi-level scatter and gather behavior. Step 1 shows the original workflow definition, with dotted lines indicating dynamic tasks that have scatter and/or follow properties yet to be resolved. In Step 2, task X scatters based on output from task W . Task Y follows X . In Step 3, the tasks Y_i scatter on output from X_i . Three versions of Step 4 are shown, depending on the follow-property of task Z . In 4a, task Z does not have the property specified. In 4b, task Z follows task X , with each Z_i gathering the output from an X_i . In 4c, task Z follows task Y , forming parallel chains of tasks.

there are also workflow-level imperatives for static input and output. These are specified in the workflow definition, and are thus independent of the task execution. However, they can be defined using expressions that depend on, for example, the output from predecessor tasks, allowing for dynamic behavior.

4 CASE STUDIES

4.1 Variant Calling Workflow

Variant calling is the process of identifying sites at which a DNA sequence of interest differs from a reference. Variants can include short sequences like single nucleotide polymorphisms (SNPs) and insertions or deletions (indels), as well as larger structural differences like inversions and translocations. The Genome Sequencing and Analysis Group at the Broad Institute in Cambridge, MA has developed a set of standardized genetic analysis pipelines that are bundled in the resource the Genome Analysis Toolkit (GATK) [28]. The toolkit includes a best practice pipeline for alignment and variant calling of short germline DNA variants that uses several different open-source software packages. We used this pipeline, in combination with more recent documentation on GATK forums and GitHub pages, to construct a SWEEP workflow for generating a variant call format (VCF) file from a single set of Illumina paired-end reads in FASTQ format.

As shown in Figure 2, the SWEEP variant calling workflow consists of 14 distinct tasks. The first one defines the set of samples of interest and forwards the information to its successors. Tasks 2-6 are used to create index files for the reference genome and its known SNPs and indels. Tasks 7-13 perform the sample-specific analysis, with Task 7 scattering over the output of Task 1. Tasks 8-13 are configured to follow Task 7, creating a parallel chain in

Figure 2: Diagram of the variant calling workflow.

the graph for each sample. The meta-information passed to Task 7 by Task 1 is propagated to its successors using the static output functionality. Task 14 gathers the results of the parallel chains and writes final output to external storage. The variant calling case study is an example of a workflow requiring computationally intensive batch processing of large amounts of data. Container-based implementations were used for Tasks 2-12, and AWS S3 was used as an intermediary data store. Figure 3 shows the DAG representation of the workflow for a problem size of 64 samples after the graph has been expanded, illustrating its parallel structure.

To evaluate the scalability of the workflow, we used individuals from the European and African superpopulations from the 1000 Genomes Project [29] AWS S3 public data set, selecting only samples with paired-end read data and excluding exome sequencing samples. We did not control for paired-end read file size or read quality when constructing this dataset, and the scalability measures were therefore subject to fluctuations due to cohort-specific characteristics. However, we note that non-uniform datasets are typical in the field of genomics. In this study, we used chromosome 20 as the reference genome due to cost and time constraints, but the workflow is easily generalizable to full genome reference sequences.

4.2 Satellite Imagery Workflow

In this case study we implemented a satellite imagery analysis pipeline to evaluate surface reflectance of lakes in interior Alaska. Northern latitudes are warming at twice the rate as the rest of the globe [30], a development that is expected to catalyze changes in environmental processes including the release of greenhouse gases. Quantifying changes in northern lakes is important as they provide crucial habitats and regulate global biogeochemical cycles, but extensive field surveys are hindered by the remoteness of the

regions. Satellite remote sensing offers a feasible mechanism for large-scale observation [31] that can improve our ability to model ecosystem dynamics in remote lakes, but fine-scale analysis comes with its own set of challenges. It requires imagery to be quickly distilled and analyzed to keep up with rapid changes over time, a process that often requires significant computational resources.

A SWEEP workflow, illustrated in Figure 4, was constructed to execute the image analysis pipeline, using Planet [32] as a provider of satellite data. Task 1 defines the geographical area of interest (AOI) and desired date range, which corresponds to problem size. These are passed to Task 2, which issues an API call to search for available imagery. Task 3 scatters over the smaller segments of interest in the satellites field of view, each task sending a clipping request to the API. These are gathered in Task 4, which sends a download request for each one, exemplifying the scatter-gather functionality. A time-sensitive link referencing the clipped image is passed to Task 5, which in turn extracts and calculates the band ratios. The final Task 6 writes the band metrics that were gathered for each lake and date to S3. Figure 5 shows the expanded task graph for a problem of size 28 days.

In many ways, this case study exemplifies emerging cloud-native workflows and their particular demands. Tasks that interact with an external API are subject to restrictions since API services often impose limits on request rate or number of concurrent connections to ensure that the service is not exploited. For this reason, the limit on the number of concurrent tasks in SWEEP was set to 7, a number that was found to be suitable by experimentation. Additionally, as the delivery and activation of Planet API services are asynchronous in nature, SWEEP must be aware of such external delays during workflow orchestration. For example, Task 3 is configured to expect a delay since the imagery activation request can take several minutes to complete. Because task errors can occur from exceeding rate limits or timeouts when connecting to the API endpoint, all tasks in the workflow are configured to use a retry mechanism with exponential back-off. The satellite imagery workflow contains function-based tasks only.

5 EVALUATION

The performance of SWEEP was evaluated on different problem sizes for the two workflows described in Section 4. The variant calling workflow was executed for 1, 2, 4, 8, 16, 32, 64 and 128 samples, and the satellite imagery workflow for 1 to 56 days in week-long increments. The workflows were run 5 times sequentially for each problem size, and we report averages over these replications unless otherwise stated.

Workflow runtime is defined as the wall time from start to the time the last task finishes. Task runtime is divided into execution time, i.e. the task duration reported by the cloud provider, and overhead. Execution times may vary because of computational load, but also as a result of variability in service due to the use of shared compute infrastructure as opposed to dedicated resources. Overhead includes delays due to failures and retries, as well as the time taken by the WMS to register changes in task status.

For cost reporting, we consider the compute charges of all tasks in a workflow, calculated based on the resource usage reported by AWS. Pricing of the US West region as of August 2019 was used,

Figure 3: DAG representation of a finished run of the variant calling workflow for 64 samples.

Figure 4: Diagram of the satellite imagery workflow.

task	memory (GB)	cpu (vCPU)	problem size (number of samples)							
			1	2	4	8	16	32	64	128
1*	.1	-	1	1	1	1	1	1	1	1
2	2	1	1	1	1	1	1	1	1	1
3	2	1	1	1	1	1	1	1	1	1
4	2	1	1	1	1	1	1	1	1	1
5	2	1	1	1	1	1	1	1	1	1
6	2	1	1	1	1	1	1	1	1	1
7	8	4	1	2	4	8	16	32	64	128
8	2	1	1	2	4	8	16	32	64	128
9	2	1	1	2	4	8	16	32	64	128
10	2	1	1	2	4	8	16	32	64	128
11	2	1	1	2	4	8	16	32	64	128
12	2	1	1	2	4	8	16	32	64	128
13*	.1	-	1	2	4	8	16	32	64	128
14*	.1	-	1	1	1	1	1	1	1	1

Table 1: Configured task memory and CPU sizes for tasks in the variant calling workflow, with number of task instances spawned for different problem sizes. Asterisks indicate function-based tasks.

ignoring free-tier allowances. Additional costs like logging and DNS services were excluded, as the compute charges constitute the largest part of the total and provide a figure that is easier to interpret and compare. For this reason, costs of remote storage are also excluded. The SWEEP server was hosted on an AWS EC2 instance of type t2.medium, the cost of which is also not included. Tables 1 and 2 show the configured memory and CPU resources for each task in the two workflows, as well as the number of tasks spawned for each problem size.

5.1 Variant Calling Workflow

Figure 6A illustrates the workflow process for a launch of 32 samples, showing execution times and overhead. For this workflow, the overheads are low compared to total task duration, and therefore

task	memory (GB)	problem size (number of days)								
		1	7	14	21	28	35	42	49	56
1	.1	1	1	1	1	1	1	1	1	1
2	.3	7	7	7	7	7	7	7	7	7
3	.1	25	50	96	148	244	260	297	319	420
4	2	1	1	1	1	1	1	1	1	1
5	.1	25	47	93	140	230	245	276	300	395
6	.1	1	1	1	1	1	1	1	1	1

Table 2: Configured task memory sizes for tasks in the satellite imagery workflow, with number of task instances spawned for different problem sizes. The number of Task 5 instances depends on how many failures there were among Task 3 instances. The numbers displayed here are examples from performed runs.

barely visible. Durations of the overhead for different problem sizes are visible in Figure 6C, and show that although the averages remain stable throughout, outliers with overhead up to 175 seconds emerge for the larger problem sizes. We observed that this was mainly due to throttling of the AWS API when launching many container-based tasks simultaneously, resulting in failures and retries.

Figure 6B shows the execution times per task type for different problem sizes. It should be noted that since the tasks operate on different samples, there is inherent variation in execution times, and that there are samples that consistently behave as outliers with runtimes over twice the average. For this reason, we consider median execution time per task type for a run, and plot the average over different runs. For most tasks, the median execution time was unaffected by problem size. Taking into account the fact that the larger problem sizes contained increasingly time-consuming samples, no significant deterioration in performance as a result of increased parallel workload was observed.

Figure 7 shows the average workflow runtime and compute cost per problem size. The fact that the cost does not vary greatly over runs for the same problem size further suggests a consistent service level, with stable execution times over different invocations. A challenging aspect of the variant calling workflow is the high variability in execution time among instances of Task 7. In traditional compute infrastructures, such workflows can be difficult to manage as they may require careful scheduling to avoid over-provisioning. A benefit of the serverless computational model is that this is inherently handled by the provider, visible in Figure 7 as a linearly rising cost even though increasingly longer-running samples are included (as indicated by the step-wise increase of workflow runtime).

Figure 5: DAG representation of a finished run of the satellite imagery workflow for 28 days. Gray color indicates failed tasks.

Figure 6: Evaluation of the variant calling workflow. A: Execution time and overhead for tasks in a run of 32 samples, ordered by starting time relative to the beginning of the workflow run. Overhead is shown in red (although barely visible for this workflow). B: Median execution time, separated by task type, and averaged over 5 runs for each problem size. C: Size of task overhead for all tasks in all runs for a given problem size.

5.2 Satellite Imagery Workflow

The satellite imagery workflow comprises function-based tasks only. Compared to the variant calling case study, these have lower execution times, and the workflow structure differs by the existence of the intermediary gathering Task 4. The parallel execution of the workflow is further limited by constraints imposed by external services. As explained in Section 4.2, due to rate limits of the Planet API, a limit of 7 parallel SWEEP tasks is imposed, and delays are in place to handle asynchronous API requests.

Figure 8A shows task execution times and overheads for a run of 7 days. In this case, the overhead does not include the time spent waiting due to the limit in task concurrency. The very high

Figure 7: Average total workflow runtime and compute cost for different problem sizes of the variant calling workflow, with shaded regions indicating standard deviation.

overheads for Task 3 instances are largely due to the high failure rate of requests to the Planet API. Retries of API requests occur within a task and are visible as variations in execution time. When a task is retried by the SWEEP engine, the extra time taken is registered as task overhead. After the maximum number of SWEEP retries is reached, the task fails and becomes a dangling node in the task graph, indicated as gray nodes in Figure 5. Overhead duration for Task 3 is shown in Figure 8B. Although there is higher variation for larger problem sizes, the average remains stable, illustrating how task concurrency can be used to tailor workflow management behavior to handle limitations imposed by external services.

Figure 8C shows the per-task execution times over all launches and problem sizes. Tasks 2 and 4 are excluded as their workloads increase with problem size. Tasks 1, 3 and 5 make requests to the Planet API, explaining much of the seen variation, as previously discussed. Task 6 is independent of external services, and shows a high level of consistency in execution time.

Figure 9 shows the average workflow runtime and compute cost for different problem sizes of the satellite imagery workflow. With the task concurrency limit, the roughly linear increase in runtime is expected, and a corresponding behavior is seen in the cost. The compute cost does not exceed \$0.03 per workflow, and exhibits a high level of consistency over different runs. An increased failure rate for requests to the Planet API is a likely explanation for the higher variation in cost for larger problem sizes.

6 DISCUSSION AND FUTURE WORK

The presented case studies exemplify the use of SWEEP for the creation of scalable workflows, but we note that there are several

Figure 8: Evaluation of the satellite imagery workflow. A: Execution time and overhead for the tasks in a run of 7 days, ordered by starting time relative to the beginning of the workflow run. Overhead is shown in red and does not include the time spent waiting due to the limit in task concurrency. **B:** Overhead sizes for all instances of Task 3 in all runs for a given problem size. **C:** Execution times over all runs and problem sizes for tasks 1, 3, 5 and 6.

Figure 9: Average total workflow runtime and compute cost for different problem sizes of the satellite imagery workflow, with shaded regions indicating standard deviation.

aspects of the serverless model that must be taken into account in the context of executing scientific workflows.

Firstly, limitations of FaaS and CaaS services are in place for most providers, and may impose constraints that render some workflows impractical or impossible to define. For example, Lambda functions have a timeout of 900 s and a maximum of 3 GB memory. Fargate maximum allocations are 4 vCPUs, 30 GB memory and 10 GB disk, with an additional 4 GB for volume mounts. However, we note that the transition away from monolithic architectures towards distributed computing has already triggered re-definition of workflows to smaller segments that are easily parallelizable. An example from genomics is described in [33], in which a variant calling pipeline

is defined that parallelizes over populations and chromosomes. As the use of serverless architectures to execute data-intensive and computationally expensive applications increases, providers may also adapt their services accordingly.

Furthermore, a consequence of the lack of control over provisioning the computational environment is that stability and response is governed by the provider’s service level agreement (SLA) and may be subject to variability. Although we did not observe high variation in execution times, overhead due to AWS API throttling were visible from only 64 parallel Fargate tasks. Both Lambda and Fargate also have limits on the number of parallel invocations in place. These were not reached by the experiments performed in this study, but would lead to additional overhead for workflows with larger parallel workloads. We plan to develop alternatives for overcoming such limitations by exploring the use of scheduling techniques that leverage multiple clouds, and making use of data from previous invocations to predict and optimize cost as well as execution time. As has been explored in [23] and [34], FaaS tasks are also subject to invocation overhead due to latency caused by the initialization process of the container that serves the function. We plan to analyze such invocation overhead of the FaaS and CaaS services used, and explore whether options like preemptive spawning of tasks can be used to obtain more consistent performance.

The stateless nature of serverless tasks also requires careful consideration of the effects of remote storage on I/O throughput and cost. In the current version of SWEEP, it is up to the user to define tasks so that they maintain data persistence in an efficient manner, but we plan to include a default solution that provides intermediary storage for user-defined tasks via an interface.

Finally, although pricing of serverless compute is higher than on-demand resources, the elasticity of FaaS and CaaS renders them more cost-effective under certain conditions. For a simple example comparing costs between Fargate and EC2, we consider a cluster of 20 m5a.xlarge instances, totalling 80 vCPUs and 320 GB memory. For an image that requires 4 vCPUs and 16 GB memory, 20 instances at maximum can be deployed on the cluster. Denoting the level of utilization as the percentage of this number that are actually deployed at a given time, we compare the cluster cost to that of running the same amount of containers on Fargate. The level at which Fargate costs exceed cluster is around 15 instances, meaning that at levels lower than 75% cluster utilization, using Fargate results in lower costs.

We acknowledge that this simple calculation does not take into account the lower costs of reserved instances and that cluster auto-scaling can be used to mitigate under-provisioning, but it illustrates the fact that the added costs of serverless are of significance only if efficient cluster utilization can be maintained. On-demand resources thus tend to be the most economical for applications with stable and predictable workloads, and we argue that for many scientific workflow scenarios, the greater elasticity outweighs the additional costs of serverless compute. As previously discussed, much of the overhead associated with cluster configuration and management is also avoided. Another advantage is that cost breakdown can be easily made at the workflow and even task level, giving very fine granularity of total charges. Per-task resource usage over time is currently made available via the SWEEP API, allowing users to fine-tune resource allocation. In the future, we plan to enhance and

automate resource optimization, and explore the tradeoff between charges for increased memory and the resulting shorter runtimes.

7 CONCLUSION

In this work we introduce SWEEP, a WMS built on the serverless execution model that allows the direct mapping of functions and containers to tasks in workflows. SWEEP's workflow definition language is designed for the paradigm of serverless workflows comprising stateless tasks and provides powerful constructs for parallelism and dynamic graph unrolling, which we demonstrate on two workflows from different scientific domains. The SWEEP system thus provides a layer of structure and orchestration on top of serverless execution frameworks, allowing a wide variety of data processing pipelines to be re-defined and executed as scalable cloud-native workflows.

ACKNOWLEDGMENTS

The authors would like to thank Microsoft AI for Earth, eScience Institute at the University of Washington for the provision of cloud credits, and Justin Ong and Meha Agarwal for their help in developing and testing the workflows.

REFERENCES

- [1] Neil Savage. Going serverless. *Communications of the ACM*, 2018.
- [2] Ioana Baldini, Kerry Chang, Perry Cheng, Stephen J. Fink, Nick Mitchell, Vinod Muthusamy, Rodric Rabbah, Aleksander Slominski, Vatche Ishakian, and Philippe Suter. Serverless Computing: Current Trends and Open Problems. In *Research Advances in Cloud Computing*, 2017.
- [3] Marc Bux and Ulf Leser. Parallelization in Scientific Workflow Management Systems. *arXiv e-prints*, page arXiv:1303.7195, Mar 2013.
- [4] Johannes Köster and Sven Rahmann. Snakemake—a scalable bioinformatics workflow engine. *Bioinformatics*, 28(19):2520–2522, 08 2012.
- [5] Michael Wilde, Mihael Hategan, Justin M. Wozniak, Ben Clifford, Daniel S. Katz, and Ian Foster. Swift: A language for distributed parallel scripting, 2011.
- [6] 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.
- [7] Bertram Ludäscher, Ilkay Altintas, Chad Berkley, Dan Higgins, Efrat Jaeger, Matthew Jones, Edward A. Lee, Jing Tao, and Yang Zhao. Scientific workflow management and the kepler system: Research articles. *Concurr. Comput. : Pract. Exper.*, 18(10):1039–1065, August 2006.
- [8] Katherine Wolstencroft, Robert Haines, Donal Fellows, Alan Williams, David Withers, Stuart Owen, Stian Soiland-Reyes, Ian Dunlop, Aleksandra Nenadic, Paul Fisher, Jiten Bhagat, Khalid Belhajjame, Finn Bacall, Alex Hardisty, Abraham Nieva de la Hidalga, Maria P. Balcazar Vargas, Shoaib Sufi, and Carole Goble. The taverna workflow suite: designing and executing workflows of web services on the desktop, web or in the cloud. *Nucleic acids research*, 41(Web Server issue):W557–W561, Jul 2013. 23640334[pmid].
- [9] Enis Afgan, Dannon Baker, Bérénice Batut, Marius van den Beek, Dave Bouvier, Martin Čech, John Chilton, Dave Clements, Nate Coraor, Björn A Grüning, Aysam Guerler, Jennifer Hillman-Jackson, Saskia Hiltmann, Vahid Jalili, Helena Rasche, Nicola Soranzo, Jeremy Goecks, James Taylor, Anton Nekrutenko, and Daniel Blankenberg. The Galaxy platform for accessible, reproducible and collaborative biomedical analyses: 2018 update. *Nucleic Acids Research*, 46(W1):W537–W544, 05 2018.
- [10] Matei Zaharia, Reynold S. Xin, Patrick Wendell, Tathagata Das, Michael Armbrust, Ankur Dave, Xiangrui Meng, Josh Rosen, Shivaram Venkataraman, Michael J. Franklin, Ali Ghodsi, Joseph Gonzalez, Scott Shenker, and Ion Stoica. Apache spark: A unified engine for big data processing. *Commun. ACM*, 59(11):56–65, October 2016.
- [11] Vinod Kumar Vavilapalli, Arun C. Murthy, Chris Douglas, Sharad Agarwal, Mahadev Konar, Robert Evans, Thomas Graves, Jason Lowe, Hitesh Shah, Siddharth Seth, Bikas Saha, Carlo Curino, Owen O'Malley, Sanjay Radia, Benjamin Reed, and Eric Baldeschwieler. Apache hadoop yarn: Yet another resource negotiator. In *Proceedings of the 4th Annual Symposium on Cloud Computing, SOCC '13*, pages 5:1–5:16, New York, NY, USA, 2013. ACM.
- [12] C. Witt, J. Dowling, M. Bux, J. Brandt and U. Leser. Hi-way: Execution of scientific workflows on hadoop yarn. In *Proceedings of the 20th International Conference on Extending Database Technology (EDBT)*, pages 668–679, 2017.
- [13] Marc Bux, Jörgen Brandt, Carsten Lipka, Kamal Hakmizadeh, Jim Dowling, and Ulf Leser. Saasfee: Scalable scientific workflow execution engine. *PVLDB*, 8:1892–1895, 2015.
- [14] HashiCorp. Terraform, 2019.
- [15] Paolo Di Tommaso, Emilio Palumbo, Maria Chatzou, Pablo Prieto, Michael L. Heuer, and Cedric Notredame. The impact of docker containers on the performance of genomic pipelines. *PeerJ*, 3:e1273–e1273, Sep 2015. 26421241[pmid].
- [16] Docker. Docker, 2019.
- [17] Paolo Di Tommaso, Maria Chatzou, Evan W. Floden, Pablo Prieto Barja, Emilio Palumbo, and Cedric Notredame. Nextflow enables reproducible computational workflows. *Nature Biotechnology*, 35(4):316–319, 2017.
- [18] Ling-Hong Hung, Jiaming Hu, Trevor Meiss, Alyssa Ingersoll, Wes Lloyd, Daniel Kristiyanto, Yuguang Xiong, Eric Sobie, and Ka Yee Yeung. Building containerized workflows using the biopot-workflow-builder (bwb). *bioRxiv*, 2018.
- [19] Jon Ander Novella, Payam Emami Khoonsari, Stephanie Herman, Daniel White-nack, Marco Capuccini, Joachim Burman, Kim Kultima, and Ola Spjuth. Container-based bioinformatics with Pachyderm. *Bioinformatics*, 35(5):839–846, 08 2018.
- [20] Alfonso Pérez, Germán Moltó, Miguel Caballer, and Amanda Calatrava. Serverless computing for container-based architectures. *Future Generation Computer Systems*, 83:50–59, 2018.
- [21] R. Qasha, J. Cala, and P. Watson. A framework for scientific workflow reproducibility in the cloud. In *2016 IEEE 12th International Conference on e-Science (e-Science)*, pages 81–90, Oct 2016.
- [22] M. Villamizar, O. Garcés, L. Ochoa, H. Castro, L. Salamanca, M. Verano, R. Casallas, S. Gil, C. Valencia, A. Zambrano, and M. Lang. Infrastructure cost comparison of running web applications in the cloud using aws lambda and monolithic and microservice architectures. In *2016 16th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (CCGrid)*, pages 179–182, May 2016.
- [23] Eric Jonas, Qifan Pu, Shivaram Venkataraman, Ion Stoica, and Benjamin Recht. Occupy the cloud: Distributed computing for the 99th. In *Proceedings of the 2017 Symposium on Cloud Computing, SoCC '17*, pages 445–451, New York, NY, USA, 2017. ACM.
- [24] Vaishaal Shankar, Karl Krauth, Qifan Pu, Eric Jonas, Shivaram Venkataraman, Ion Stoica, Benjamin Recht, and Jonathan Ragan-Kelley. numpywren: serverless linear algebra. *arXiv e-prints*, page arXiv:1810.09679, Oct 2018.
- [25] Fission. Fission workflows, 2019.
- [26] Qingye Jiang, Young Choon Lee, and Albert Y. Zomaya. Serverless execution of scientific workflows. In Michael Maximilien, Antonio Valleillo, Jianmin Wang, and Marc Oriol, editors, *Service-Oriented Computing*, pages 706–721, Cham, 2017. Springer International Publishing.
- [27] M. Orzechowski, B. Balis, K. Pawlik, M. Pawlik, and M. Malawski. Transparent deployment of scientific workflows across clouds - kubernetes approach. In *2018 IEEE/ACM International Conference on Utility and Cloud Computing Companion (UCC Companion)*, pages 9–10, Dec 2018.
- [28] Aaron McKenna, Matthew Hanna, Eric Banks, Andrey Sivachenko, Kristian Cibulskis, Andrew Kernysky, Kiran Garimella, David Altshuler, Stacey Gabriel, Mark Daly, and Mark A. DePristo. The genome analysis toolkit: A MapReduce framework for analyzing next-generation DNA sequencing data. *Genome Research*, 2010.
- [29] 1000 Genomes Project Consortium, Adam Auton, Lisa D Brooks, Richard M Durbin, Erik P Garrison, Hyun Min Kang, Jan O Korbel, Jonathan L Marchini, Shane McCarthy, Gil A McVean, and Gonçalo R Abecasis. A global reference for human genetic variation. *Nature*, 2015.
- [30] Thomas F. Stocker, Dahe Qin, Gian Kasper Plattner, Melinda M.B. Tignor, Simon K. Allen, Judith Boschung, Alexander Nauels, Yu Xia, Vincent Bex, and Pauline M. Midgley. *Climate change 2013 the physical science basis: Working Group I contribution to the fifth assessment report of the intergovernmental panel on climate change*. 2013.
- [31] Catherine Kuhn, Aline de Matos Valerio, Nick Ward, Luke Loken, Henrique Oliveira Sawakuchi, Milton Kampel, Jeffrey Richey, Philipp Stadler, John Crawford, Rob Striegl, Eric Vermote, Nima Pahlevan, and David Butman. Performance of Landsat-8 and Sentinel-2 surface reflectance products for river remote sensing retrievals of chlorophyll-a and turbidity. *Remote Sensing of Environment*, 2019.
- [32] Planet Team. Planet application program interface: In space for life on earth. San Francisco, CA, 2018.
- [33] Suyash S. Shringarpure, Andrew Carroll, Francisco M. De La Vega, and Carlos D. Bustamante. Inexpensive and highly reproducible cloud-based variant calling of 2,535 human genomes. *PloS one*, 10(6):e0129277–e0129277, Jun 2015. 26110529[pmid].
- [34] Wes Lloyd, Shruti Ramesh, Swetha Chinthalapati, Lan Ly, and Shrideep Pallickara. Serverless computing: An investigation of factors influencing microservice performance. *Proceedings - 2018 IEEE International Conference on Cloud Engineering, IC2E 2018*, pages 159–169, 2018.