Toward Scientific Workflows in a Serverless World

Aakash Khochare and Yogesh Simmhan
Indian Institute of Science, Bangalore 560012 INDIA
Email: {aakhochare, simmhan}@IISc.ac.in

Sameep Mehta and Arvind Agarwal
IBM India Research Lab, Bangalore INDIA
Email: {sameepmehta, arvagarw}@in.ibm.com

2022 IEEE 18th International Conference on e-Science (e-Science) | 978-1-6654-6124-5/22/$31.00 ©2022 IEEE | DOI: 10.1109/ESCIENCE55777.2022.00057

***Abstract*—Serverless computing and FaaS have gained popularity due to their ease of design, deployment, scaling and billing on clouds. However, when used to compose and orchestrate scientific workflows, they pose limitations due to cold starts, message indirection, vendor lock-in and lack of provenance support. Here, we propose a design for a Serverless Scientific Workflow Orchestrator that overcomes these challenges using techniques like function fusion, pilot invocations and data fabrics.**

I. Introduction

Serverless computing allows applications to be packaged, deployed and managed within clouds, typically in containers, without the developer having to configure the hosting environment or manually scaling the cloud resources at runtime [1]. *Function as a Service (FaaS)* has evolved from the micro-services pattern of decomposing an application into stateless building-block functions and executing them in an event-driven manner. The functions themselves are deployed and managed using a serverless design. FaaS has garnered popularity as it out-sources the deployment, resilience and elastic auto-scaling to the cloud service provider, allowing the developer to focus on the logic. It also offers fine-grained billing, based only on resources used during the function execution.

FaaS and serverless can democratize scientific research on the cloud [2], with early efforts on using FaaS to design and run eScience applications on clouds [3]. *Workflows* have long been popular to compose scientific applications in a modular and reusable manner, and be executed on distributed resources by workflow systems [4]. While FaaS helps function developers and cloud operators, it still poses challenges for the application developer who assembles these functions together, and on the execution model due to performance inefficiencies for pipelines. In this short paper, we attempt to answer the question: *How can scientific workflows be intuitively composed from FaaS functions, and efficiently orchestrated on the cloud?* We first identify some of the limitations of FaaS for scientific workflows (§ II), and then propose the *SerWO workflow orchestrator* design to address these deficiencies (§ III).

II. Limitations of Workflows using Serverless

In FaaS, developers provide a stateless function logic in one of a variety of languages. Each function is executed in an exclusive container or micro-VM. Functions are invoked through an API or triggered based on an event, with a cloud gateway routing the request to an available container. The cloud provider instantiates new containers and spins down idle ones, based on the request load. The event-driven model can be used to chain functions. Functions do not directly communicate with each other, instead using cloud message brokers or storage to trigger execution and transfer data.

There is limited support for composing functions into workflows or dataflows, e.g., using AWS Step Functions. Most are tightly-coupled to specific cloud environments. Common standards are still evolving, e.g., CNCF's Serverless Workflow. Scientific workflow platforms for serverless execution on public clouds have been proposed [5]. But they suffer from several of the limitations of serverless, as described below:

a) Cold Starts: In a serverless execution, a container is provisioned on-demand when a function is invoked, with a startup latency of 100s of ms. If the function is not called often, none of its containers may be active to reduce costs. For one-off or periodic scientific workflows, all function containers will be incrementally instantiated as the workflow executes, causing $\mathcal{O}(secs)$ cumulative startup latencies. Some [6] have proposed to use IaaS VMs to host short functions to prevent cold-starts, but this mitigates the benefits of serverless.

b) Message Indirection: Serverless functions are not IP-addressable since their containers are transient. Hence, point-to-point communication is not possible between adjacent functions chained in a workflow. Functions use message brokers or database triggers to coordinate a workflow, with cloud storage for larger payloads. This indirection causes 2-rounds of I/O or network communication, besides the monetary cost for the operations. This is amplified in data-intensive scientific workflows that pass large files between workflow tasks.

c) Hybrid Clouds: eScience functions may have data or hardware dependencies that tie them to specific cloud providers, or even to edge resources co-located with instruments. Some functions may require longer execution times, which exceed limits set by some FaaS providers, e.g., $900s$ for AWS Lambda. So scientific workflows may reasonably expect to execute functions in a multi-cloud environment. However, current FaaS platforms are proprietary to individual cloud providers, or limited to a single data center. Existing solutions [7] end up switching between FaaS and local HPC clusters, but fail to offer a purely serverless design.

d) Provenance Support: Data Provenance is essential for reproducibility in scientific workflows [8]. While serverless offer fine-grained billing and metrics, this needs to be annotated and coupled with the workflow orchestration to translate into meaningful provenance. This becomes even more challenging in a hybrid cloud setup, with distributed data repositories.

978-1-6654-6124-5/22/$31.00 ©2022 IEEE
DOI 10.1109/eScience55777.2022.00057

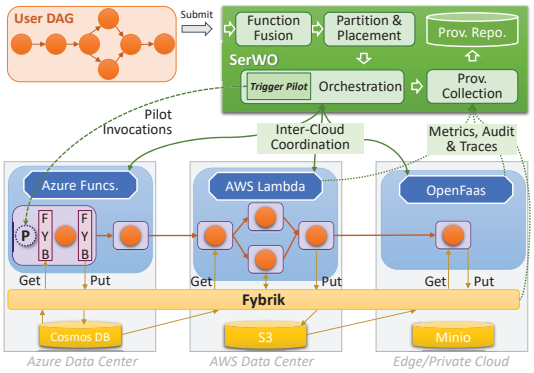

Fig. 1: Architecture of SerWO on a Hybrid Cloud

III. Serverless Workflow Orchestrator Design

The limitations that we have identified for scientific workflows executing serverless functions is not necessarily new [6], [7]. However, we advocate a pure serverless FaaS approach to addressing these limitations, thus retaining all the advantages of serverless computing. We also propose a workflow orchestrator that is agnostic to the cloud service provider – public, private or even edge – and supports a hybrid cloud deployment.

Fig. 1 shows the architecture of our proposed *Serverless scientific Workflow Orchestrator (SerWO)*. It is described below in detail. In summary, SerWO accepts a workflow as a Directed Acyclic Graph (DAG), defined using an open format like CNCF's serverless workflow specification. It then performs *function fusion* to group adjacent functions, and *partitions and deploys* sub-DAGs onto one or more FaaS providers. The placement is based on function dependencies, and on performance and cost constraints specified for the workflow. SerWO then *orchestrates* the execution of the DAG functions, potentially across data centers. Data exchange between functions happen through a multi-cloud *data fabric* such as Fybrik [9]. Before the execution of a function, the orchestrator preemptively makes *pilot invocations* to warmup downstream function containers and avoid cold-starts. Lastly, we track the execution of the task transitions, function metrics and data audit trail as part of a *provenance repository*.

a) Function Fusion: Adjacent functions in a workflow DAG can be "fused" together into a single function, and executed as a single function in one container. This can be beneficial if the function execution time is so short that the container startup overheads dominate, and the fused function has a longer execution time and hence better overhead-amortization. Secondly, fusion avoids indirect message/data passing overheads between adjacent tasks, doing an in-memory/intra-container transfer than through Fybrik.

b) Placement and Orchestration in a Hybrid Cloud: Functions with dependencies on specific data centers, or pricing arbitrage across cloud providers are considered during scheduling. We partition the functions in the DAG into sub-DAGs, based on these dependency and performance needs, such that each runs within one cloud data center or edge. Each sub-DAG is deployed to the native FaaS platform for that cloud, e.g., Step functions on AWS or OpenFaaS on a private cloud or edge. So SerWO only has to actively coordinate only at sub-DAG boundaries, with the coordination within functions of a sub-DAG managed natively. SerWO could also cache and return results of commonly executed idempotent functions. We also leverage *Fybrik* [9], an emerging data fabric platform for data orchestration across hybrid clouds. It makes data discovery and access transparent, both within the same cloud data center and across cloud providers. We automatically wrap functions with Fybrik to put and get messages/data between workflow tasks.

c) Pilot Invocations: We mitigate cold-starts by leveraging the "pilot jobs" concept, used as place-holders in HPC clusters for multi-level scheduling [10]. Rather, we use a pilot invocation, which is a warm-up NoOp execution of a downstream function in the DAG that activates at least one container for that function but does not execute its business logic. When a function A in the DAG is nearing completion, the orchestrator does a pilot invocation of its successor function B so that B's container is instantiated. It can even prefetch the output data of A from Fybrik to further hide the data transfer overheads. The pilot logic is part of our automated function wrapper. The cost of a pilot execution is negligible, but it avoids a cold-start while retaining a serverless design.

d) Data Provenance: We propose to integrate the audit logs of the data fabric, which tracks payload exchange between tasks within and across clouds, with the function execution traces provided by the FaaS platforms and store a lineage graph in a provenance repository, leveraging W3C PROV. This can also help assert if data compliance constraints were met.

The implementation of SerWO is a work in progress, and as part of future work, we propose to evaluate it against native FaaS platforms on cost and performance.

References

[1] K. Owens *et al.*, "Serverless whitepaper v1.0," Cloud Native Computing Foundation (CNCF), Tech. Rep., 2018.

[2] G. C. Fox, V. Ishakian, V. Muthusamy, and A. Slominski, "Status of serverless computing and function-as-a-service (faas) in industry and research," Tech. Rep., 2017.

[3] R. Chard, Y. Babuji, Z. Li, T. Skluzacek, A. Woodard, B. Blaiszik, I. Foster, and K. Chard, "Funcx: A federated function serving fabric for science," in *International Symposium on High-performance Parallel and Distributed Computing (HPDC)*, 2020, pp. 65–76.

[4] E. Deelman, D. Gannon, M. Shields, and I. Taylor, "Workflows and e-science: An overview of workflow system features and capabilities," *Future Generation Computer Systems (FGCS)*, vol. 25, no. 5, 2009.

[5] 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 (FGCS)*, vol. 110, 2020.

[6] R. B. Roy, T. Patel, V. Gadepally, and D. Tiwari, "Mashup: making serverless computing useful for hpc workflows via hybrid execution," in *ACM Symp. on Princ. and Practice of Parallel Prog. (PPoPP)*, 2022.

[7] Q. Jiang, Y. C. Lee, and A. Y. Zomaya, "Serverless execution of scientific workflows," in *International Conference on Service-Oriented Computing (ICSOC)*. Springer, 2017.

[8] Y. L. Simmhan, B. Plale, and D. Gannon, "A survey of data provenance in e-science," *ACM Sigmod Record*, vol. 34, no. 3, 2005.

[9] M. Factor and R. Kat, "Fybrik: A cloud-native platform to control data usage," IBM Research, Tech. Rep., 2021.

[10] M. Turilli, M. Santcroos, and S. Jha, "A comprehensive perspective on pilot-job systems," *ACM Computing Surveys (CSUR)*, vol. 51, 2018.