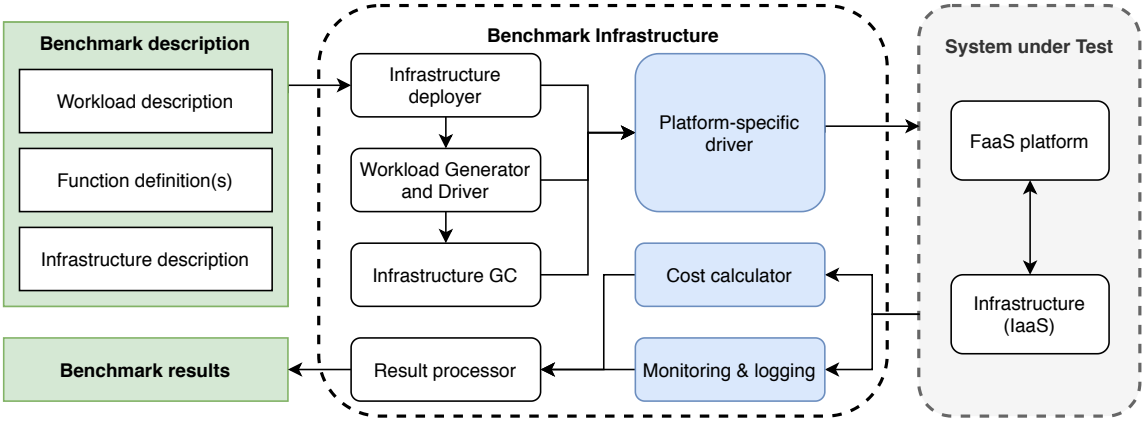

Figure 2: Overview of the serverless benchmark architecture. The rectangular green boxes indicate the static artifacts necessary for and generated by the benchmark, and the blue rounded boxes indicate the platform-specific components.

Each benchmark is represented by a *benchmark description*, which consists of a workload, function deployment, and infrastructure description. Based on this description, first the *infrastructure deployer* uses the *infrastructure description* and the *function definitions* to configure the cloud *infrastructure* and (optionally) orchestrate the deployment of a self-managed *FaaS platform*. Once the infrastructure is ready, the *workload generator and driver* pushes workload to the system according to the *workload description*. After the benchmark has been completed, the *benchmark GC* will ensure that the benchmark resources are pruned from the cloud environment.

The *monitoring & logging* component collects metrics during a benchmark run. This component is also platform-specific, as the way monitoring data is stored and retrieved differs widely between platforms. Alongside, the *cost calculator* retrieves and calculates the cost that has been incurred during the benchmark execution. Together the monitoring and cost data are post-processed by a *result processor* and stored as the final *benchmark results*.

3.4 Implementation

The development of the benchmark remains an ongoing process. However, we can already share highlights on the current status of the implementation of the benchmark.

We use Go for the framework, due to its focus on networking and its popularity within the (distributed) systems community. For most experiments, we use NodeJS or Python for the serverless functions, as they are widely supported by existing serverless platforms [10].

The field of serverless computing evolves rapidly, so—without updates—benchmark results become outdated quickly. For this reason, we focus on the implementation of reproducibility and workflow automation. This will allow us to routinely rerun the benchmark and maintain an overview of up-to-date results.

Our goal is to make the benchmark, experiments, and results open-source.[5] Moreover, beyond making the project open-source, we specifically aim for developer experience; it should be straightforward to deploy and run the benchmark, as well as easy to contribute to. With these aims, we hope to foster a long-lived, community-wide serverless benchmark.

3.5 Ideas for the longer-term future

Finally, there are several topics which we deem interesting yet out-of-scope for the near-future of the current project.

First, we expect that privacy and other data regulations, such as GDPR,[6] will increasingly affect the performance characteristics of cloud computing. For serverless computing specifically, the dynamic and ephemeral nature makes it an appropriate model for privacy-sensitive workloads, allowing platforms to schedule the execution on a granular level.

Second, as serverless computing becomes increasingly relevant for data-intensive workloads—e.g., in graph processing [9] and object storage [8]—additional experiments targeting these kinds workloads will be needed. A benchmark will need to evaluate the (possible) interplay between the storage and execution platforms, and explore how data-intensive serverless applications are designed.

4 PRELIMINARY RESULTS

We have run preliminary experiments before and during the development of our work-in-progress benchmark. The results of these experiments serve as an initial validation of the benchmark design.

We describe the results of two real-world experiments we have performed on closed- and open-source serverless platforms: (1) a basic evaluation of event propagation using a state-of-the-art message queue as proxy for a event management system in a cloud, and (2) an evaluation of the performance and cost of several FaaS platforms running the same workload.

4.1 Event propagation

The goal of the preliminary evaluation of the event propagation was to explore the performance difference between event management system configurations and to serve as a reference for the results of the different cloud providers.

[5] Due to the prototype state of the benchmark, it is currently closed source.

[6] https://eur-lex.europa.eu/eli/reg/2016/679/oj

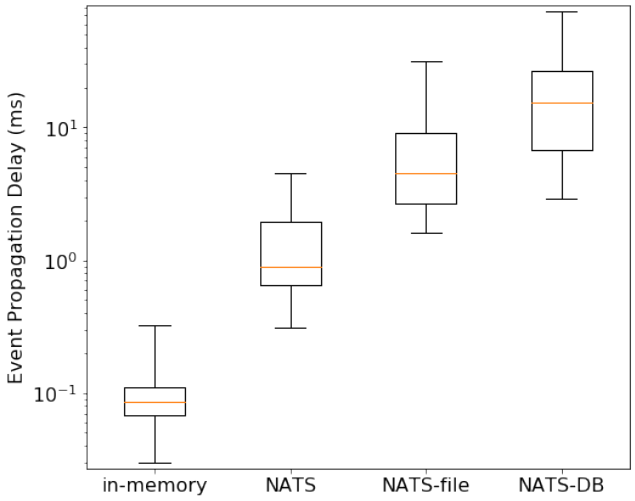

Figure 3: The impact of different event-queue configurations on the delay of event propagation.

For the event management system, we chose to evaluate configurations of an open-source, state-of-the-art message queue. These systems are typically used for event management throughout complex ecosystems (such as clouds), and can therefore be viewed as a reasonable proxy. We settled on evaluating configurations of NATS Streaming,[7] which is comparable in functionality and performance to other state-of-the-art alternatives, such as RabbitMQ and Kafka. The configurations we evaluate are the modes of persistence, which specify how the messages (or events) are persisted. We choose this parameter because it is unknown what persistence model is used within the major clouds, and it was technically feasible for a time-constrained preliminary experiment.

For this *real-world experiment*, we used a setup of a single driver VM (8 Intel Xeon CPUs, 32 GB RAM, 512 GB SSD) submitting a workload to a single VM (4 Intel Xeon CPUs, 16 GB RAM, 256 GB SSD) operating a NATS Streaming server. The VMs were deployed in the same datacenter, connected to each other by a 1 Gbit/s Ethernet link. The event propagation delay is the duration between the driver sending the message and the driver receiving the same message while subscribed to the message queue.

For the workload we used the 10-minute Chronos trace [6]. This workload trace is an ETL workload, which originates from an industrial process use case. It consists of submissions of a 3-task workflow, with on average 5.4 tasks being submitted per second with peaks of 22 events per second. For the event propagation experiment, we assumed that each task maps to one event.

Figure 3 shows the impact of the configurations of the message queue on the event propagation. As a baseline, *in-memory* skips the entire message queue and immediately returns the result to the subscriber. The *NATS* configuration is the default configuration of the system, which stores messages solely in-memory. *NATS-file* stores the messages on the local filesystem (SSD). The *NATS-DB* uses a SQL database—Postgres in this case—to persist the messages.

[7]https://github.com/nats-io/nats-streaming-server

In the context of serverless functions, the difference between these delays would likely have a significant impact on the performance (see Figure 4). Since there are numerous parameters to configure for these types of systems, these results hint that the internally-built event management systems in the major clouds will differ in performance.

4.2 Function runtime overhead

In the second experiment, our goal was to perform an initial evaluation of the overall runtime overhead of several serverless platforms. Although this type of experiment has been included in some existing benchmarks, we additionally focused on exploring the evaluation of an open-source FaaS platform and workflow orchestration systems.

We evaluated the serverless platforms of the three major cloud providers (Azure, AWS, and Google), and a state-of-the-art, open-source FaaS platform, called Fission.[8] As a baseline, we additionally include SimFaaS,[9] which is a simple FaaS simulator.

For the managed FaaS platform, we evaluated AWS Lambda, Google Cloud Functions, and Azure Functions. For each of the platforms, we used similar configurations (e.g., 128 MB RAM) and deployed the driver machine (1 CPU, 4 GB RAM) in the same region as the functions. For this experiment we again used the Chronos workload, described in the event propagation experiment. We used the function execution runtimes reported by the FaaS platforms themselves to eliminate network latency impact—although this does require us to trust the self-reported metrics of the platforms. The costs of the workloads are calculated post facto, ignoring any promotional pricing. The costs are separated into costs directly attributable to the FaaS function (*function execution costs*) and costs related to the orchestration of the Chronos workflows (*workflow orchestration costs*). For the self-deployed platform (Fission), we calculated the costs (*self-management costs*) by amortizing the total machine costs incurred during the experiment over the Chronos runs. The runtime overhead is calculated by subtracting the minimum task runtime from the actual time spend on the executing task. We deployed Fission on a cluster of 3 small VMs (1 CPU, 3.75 GB RAM), and SimFaaS on a larger VM (1 CPU, 8 GB RAM).

In Figure 4 the boxplots show that from the managed serverless platforms, Google has the lowest overhead in this experiment, achieving overheads between 50 ms and 18 ms. AWS Lambda and Azure functions have slightly higher overheads, but lower performance variability.

Despite the preliminary nature of these results, we observe that there are cases in which the performance of the serverless platforms of major clouds can differ wildly for the same workload. The costs show similar variability in the costs per platform (see Figure 5). The results also highlight the challenge of comparing self-deployed serverless platforms to managed serverless platforms; although Fission technically costs significantly less than the managed platforms for equal performance, it does not account for the cost of operating these self-deployed platforms. We consider exploring how to compare these approaches fairly a key part of our future work.

[8]https://github.com/fission/fission
[9]https://github.com/erwinvaneyk/simfaas

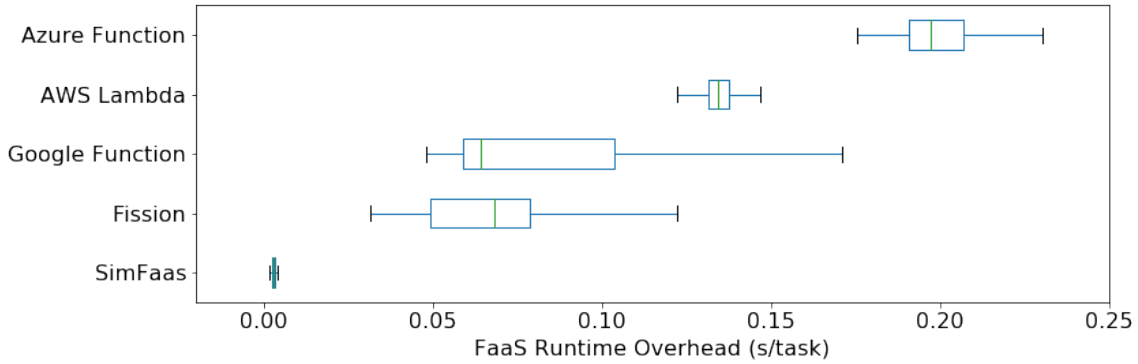

Figure 4: The performance overhead of FaaS platforms when executing the Chronos workload.

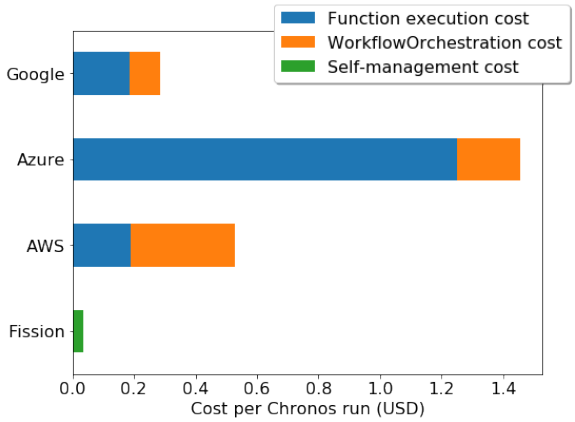

Figure 5: The cost of running the same workload, Chronos (see text), on different serverless platforms.

5 CONCLUSION

The increasingly popular serverless industry, and especially its function-based approach (FaaS), is based on emerging technology. Learning from history, we argued for the need to develop comprehensive benchmarking tools for serverless and FaaS technology.

We envisioned a benchmark designed with a structured, principled approach, aiming to evaluate the performance of the typical components of serverless platforms. Going beyond microbenchmarks, the benchmark evaluates the broader serverless ecosystem, the events that trigger functions, the overhead introduced by fetching function code, and how realistic serverless applications use the platforms. We also presented preliminary, real-world, experimental results across several closed- and open-source serverless platforms.

Our future work consists broadly of two stages. The first stage is to complete the benchmark implementation and perform a comprehensive investigation of managed serverless platforms. In the second stage, we will iterate on the existing benchmark, further analyze self-deployed serverless platforms, and evaluate the performance of more complex applications in more data-intensive domains—such as machine learning, and graph processing.

Finally, we want to end this vision with a call to action: this benchmark already is a collaborative effort of universities across the globe, and we invite the community to join this effort.

ACKNOWLEDGMENTS

The work presented in this article has benefited from discussions within the SPEC-RG Cloud Group, and further in the Cloud Control Workshop series organized by Erik Elmroth and his team.

REFERENCES

[1] Erwin Van Eyk, Alexandru Iosup, Cristina L. Abad, Johannes Grohmann, and Simon Eismann. 2018. A SPEC RG Cloud Group's Vision on the Performance Challenges of FaaS Cloud Architectures. In *ICPE WS 2018*. 21–24.

[2] Kamil Figiela, Adam Gajek, Adam Zima, Beata Obrok, and Maciej Malawski. 2018. Performance evaluation of heterogeneous cloud functions. *Concurrency and Computation: Practice and Experience* 30, 23 (2018).

[3] Joseph M. Hellerstein, Jose M. Faleiro, Joseph Gonzalez, Johann Schleier-Smith, Vikram Sreekanti, Alexey Tumanov, and Chenggang Wu. 2019. Serverless Computing: One Step Forward, Two Steps Back. In *9th Biennial Conference on Innovative Data Systems Research (CIDR)*.

[4] Eric Jonas et al. 2019. Cloud Programming Simplified: A Berkeley View on Serverless Computing. *CoRR* abs/1902.03383 (2019). arXiv:1902.03383 http://arxiv.org/abs/1902.03383

[5] Hyungro Lee, Kumar Satyam, and Geoffrey C Fox. 2018. Evaluation of Production Serverless Computing Environments. In *Third International Workshop on Serverless Computing (WoSC)*.

[6] Shenjun Ma, Alexey Ilyushkin, Alexander Stegehuis, and Alexandru Iosup. 2017. Ananke: A q-learning-based portfolio scheduler for complex industrial workflows. In *2017 IEEE International Conference on Autonomic Computing (ICAC)*. IEEE.

[7] G. McGrath and P. R. Brenner. 2017. Serverless Computing: Design, Implementation, and Performance. In *2017 IEEE 37th International Conference on Distributed Computing Systems Workshops (ICDCSW)*. 405–410.

[8] Josep Sampé, Marc Sánchez-Artigas, Pedro García-López, and Gerard París. 2017. Data-driven serverless functions for object storage. In *Proceedings of the 18th ACM/IFIP/USENIX Middleware Conference*. 121–133.

[9] Lucian Toader, Alexandru Uta, Ahmed Musaafir, and Alexandru Iosup. 2019. Graphless: Toward serverless graph processing. In *2019 18th International Symposium on Parallel and Distributed Computing (ISPDC)*. IEEE, 66–73.

[10] Erwin van Eyk, Johannes Grohmann, Simon Eismann, André Bauer, Laurens Versluis, Lucian Toader, Norbert Schmitt, Nikolas Herbst, Cristina Abad, and Alexandru Iosup. 2019. The SPEC-RG Reference Architecture for FaaS: From Microservices and Containers to Serverless Platforms. *IEEE Internet Comp.* (2019).

[11] Erwin van Eyk, Alexandru Iosup, Simon Seif, and Markus Thömmes. 2017. The SPEC cloud group's research vision on FaaS and serverless architectures. In *Proceedings of the 2nd International Workshop on Serverless Computing*. 1–4.

[12] Erwin van Eyk, Lucian Toader, Sacheendra Talluri, Laurens Versluis, Alexandru Uță, and Alexandru Iosup. 2018. Serverless is more: From PaaS to present cloud computing. *IEEE Internet Computing* 22, 5 (2018), 8–17.

[13] Liang Wang, Mengyuan Li, Yinqian Zhang, Thomas Ristenpart, and Michael Swift. 2018. Peeking Behind the Curtains of Serverless Platforms. In *2018 USENIX Annual Technical Conference (USENIX ATC 18)*. USENIX Association, Boston, MA, 133–146. https://www.usenix.org/conference/atc18/presentation/wang-liang

[14] CNCF Serverless WG. 2018. CNCF WG-Serverless Whitepaper v1.0. https://github.com/cncf/wg-serverless/blob/master/whitepaper/cncf_serverless_whitepaper_v1.0.pdf. Accessed 2018-07-29.