An Interactive Analysis of Serverless Cloud Infrastructure

Thomas Bodner
Hasso Plattner Institute, University of Potsdam
Potsdam, Germany
thomas.bodner@hpi.de

Tilmann Rabl
Hasso Plattner Institute, University of Potsdam
Potsdam, Germany
tilmann.rabl@hpi.de

ABSTRACT

Data processing systems are increasingly being deployed in the cloud, because of the benefits of elasiticity and short-term resource provisioning. In recent years, serverless cloud computing is offered in the form of highly elastic resource pools. However, analyzing and understanding the performance and cost characteristics of serverless cloud infrastructure in the context of data processing is challenging. In this demonstration, we present our Skyrise evaluation framework for experimentation in serverless data processing. The framework provides a suite of micro-benchmarks and a serverless query engine to run end-to-end workloads. Users can benchmark serverless compute and storage resources and observe the impact of their performance on data-intensive applications. In addition, they can study the trade-offs compared to server-based systems. Utilizing the inherent elasticity of serverless resources, our framework facilitates interactive analysis and supports the understanding of key characteristics of serverless data processing for data system engineers.

1 INTRODUCTION

Serverless infrastructure relieves users of the provisioning and management of servers [16]. Services such as AWS Lambda [6] and S3 [5] allocate fine-grained resources based on user consumption. They provide more elasticity than conventional cloud infrastructure and promise cost-effectiveness for sporadic usage. Although data systems benefit from the elasticity of cloud infrastructure [10, 15] and aim for finer-grained elasticity [18, 24], they have not adopted serverless cloud resources. There is a lack of understanding of whether and when serverless infrastructure is a viable foundation for data processing. In particular, the impact of networking and storage performance requires better understanding [19–21]. Additionally, the economic trade-offs of employing serverless resources need more attention.

In this demonstration, we present the *Skyrise* framework for evaluating serverless infrastructure for data processing workloads. Skyrise provides a suite of micro-benchmarks for various serverless infrastructure services and a serverless query engine to run application workloads. Thus, the framework facilitates a comprehensive analysis of the performance and cost factors of serverless data processing throughout the stack. The framework currently supports running experiments on the widely used AWS serverless infrastructure [9]. This includes the Lambda function as a service (FaaS) platform [6], and the storage services S3 [5], DynamoDB [1], and EFS [4]. Micro-benchmarks measure key performance metrics of the compute and storage resources, such as function startup time and network throughput, as well as storage throughput, operations per second (IOPS), and latency. For the evaluation of application performance, Skyrise's query execution engine supports a set of predefined queries from the TPC-H and TPCx-BB benchmarks. Query result metrics comprise the wall-clock and the cumulative compute times, and the cost of any used cloud resources.

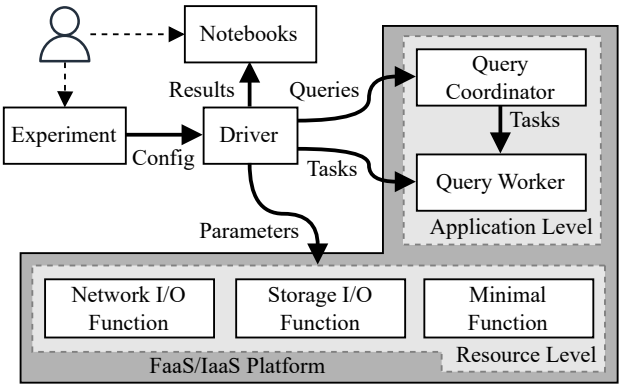

Figure 1: Architecture of the Skyrise evaluation framework showing the experiment execution flow from config to plot.

During this demonstration, we let users (1) run experiments showing characteristic features of serverless cloud infrastructure and (2) observe how these translate to application performance. The results are summarized in Jupyter notebooks to (3) aid in comprehension and comparison with server-based applications. We invite users to explore (4) different Skyrise benchmarks and benchmark configurations.

We summarize our demonstration as follows:

(1) We evaluate serverless compute and storage infrastructure for large-scale, I/O-heavy workloads both at the resource and application levels.
(2) We observe the impact of key performance characteristics of serverless infrastructure on data-intensive applications.
(3) We analyze the trade-offs between serverless and server-based systems with respect to performance and economics.
(4) Skyrise is open-source.[1] We invite the attendees to explore its broader set of benchmarks and to extend it for further experiments and cloud infrastructure services.

The rest of the paper is structured as follows. In Section 2, we introduce the Skyrise evaluation framework. In Section 3, we present our demonstration scenarios. We conclude in Section 4.

2 THE SKYRISE EVALUATION FRAMEWORK

Analyzing infrastructural components from scratch is tedious. It not only requires implementing benchmarks against the APIs of cloud providers but also requires the automation of resource provisioning, data preparation, experiment configuration and execution, as well as result processing. To enable the evaluation of serverless infrastructure, we develop the Skyrise evaluation framework [13]. Our framework includes a comprehensive suite of micro-benchmarks for serverless resources and integrates a

[1] Skyrise is available at https://github.com/hpides/skyrise.

Table 1: Overview of experiment configurations.

System under Test	Functions	Parameters	Metrics
Lambda, EC2	Minimal, Network, Storage I/O	Instance Size & Count	I/O Throughput, Startup Latency, Idle Lifetime
S3, DynamoDB, EFS	Storage I/O	File Size & Count	I/O Throughput, IOPS, Latency
Skyrise Query Engine	Query Coordinator & Worker	Queries, Scale Factor, Deployment Mode	Query Latency & Cost

full-fledged serverless query execution engine [11, 14]. The query engine allows to understand how resource-level effects translate to application performance. The coordinator and worker nodes are deployed as serverless cloud functions and employ serverless storage to load inputs and communicate outputs. As a deployment alternative, the query engine runs on virtual servers [2], enabling the comparison with server-based execution.

Adding and comparing more serverless query engines requires a thin wrapper to execute queries on these systems and extract runtimes and costs. We integrate our framework with the AWS compute services Lambda and EC2, as well as the data storage services S3, DynamoDB, and EFS. While we employ the cloud infrastructure services and C++ SDK of AWS, there are equivalent services, APIs, and SDKs in Microsoft Azure and Google Cloud Platform. Supporting other cloud platforms does not require any architectural changes. The current set of micro-benchmarks allows the evaluation, e.g., of compute unit startup times, network, and storage performance. Extending our framework with more benchmarks is done via a conventional interface with functions including setup and teardown routines, workload generation, and result processing.

Experiment Flow. The framework supports experiments on two levels of the stack. On the resource level, the framework employs micro-benchmarks measuring performance metrics of compute and storage services. On the application level, i.e., complete queries, the framework uses the query engine.

To execute experiments, the framework deploys and invokes cloud function binaries. Figure 1 gives an overview of this process. Each experiment defines a configuration, which is submitted to a driver. Depending on the experiment level, the driver invokes a specific function binary. For resource-level experiments, the driver invokes one of the micro-benchmark functions with a set of basic parameters, such as the S3 bucket or DynamoDB table. For application-level experiments, the driver executes queries with the query engine by calling the query coordinator function, which in turn breaks up queries into tasks and schedules worker functions for them. Table 1 gives an overview of supported experiment configurations. For configurations targeting EC2, we provide a shim layer that resembles the Lambda execution environment to run function binaries on servers.

When the experiment ends, the driver receives result metrics from multiple sources: Logs, traces, and a response from the invoked function. For resource-level experiments, the metrics include, e.g., timestamps, request counts, latencies, and throughputs. The driver then aggregates these results and estimates the experiment cost using the AWS price list service, disregarding any bulk discounts. For application-level experiments, the query coordinator function returns high-level metrics, such as query latency and cost. Finally, the driver stores the results in a JSON file, which is consumed by one of our Jupyter notebooks [17] for summary and visualization.

Interactive Analysis. The elasticity of serverless resources enables the interactive analysis of many of their characterisics. Serverless functions start up in milliseconds and exhibit their burstable performance at the beginning of their lifetime. Serverless storage is always available to take user requests.

In our suite of Jupyter notebooks, users can step through the experiments, exploring various configuration parameters. Re-executing steps takes seconds to minutes (for terabyte-scale queries) and allows to build an intuition for serverless cloud infrastructure. This is complemented by tooling to debug and profile our query engine [12] and the AWS web UI [7].

3 DEMONSTRATION SCENARIOS

Our demonstration employs Skyrise to run experiments against the serverless compute and storage services of AWS. The attendees take on the role of a data system engineer using our suite of Jupyter notebooks on a local laptop to progress through the experiments step by step. They inspect the (intermediate) results to observe key features of serverless infrastructure and their impact on data processing systems to inform system design.

In particular, we demonstrate how burstable network throughput in serverless functions impacts scan-heavy queries. We then show how shuffle queries benefit from IOPS scaling in object storage. Finally, we compare the performance and cost of query execution on both serverless and server-based compute resources.

In the following three scenarios, we use the TPC-H [22] and TPCx-BB [23] datasets with varying scale factors from 1 to 1,000. The dataset tables are split into Parquet files using ZSTD compression and are stored in AWS S3. We employ the standard data generators and do not partition or sort on any specific keys. We deploy the function binaries on ARM-based Lambda functions [8] and virtual servers of the EC2 C6g family [3]. Our notebooks and framework driver have low resource requirements and run on a regular laptop machine on site at the conference venue.

3.1 Network Bursting for Scan-heavy Queries

Our first scenario illustrates the burstable network throughput of serverless functions and how it can be exploited for scan-heavy queries. Functions start with an increased (burst) network throughput that depletes and recharges over time.

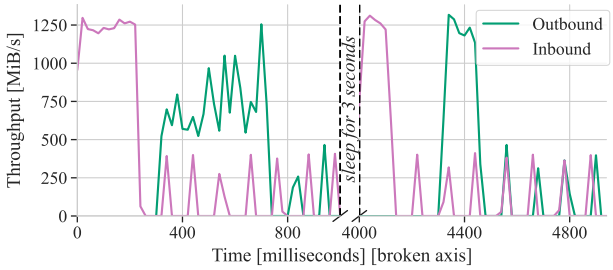

Figure 2: Serverless compute function network throughput with short sleep to refill inbound/outbound burst budgets.

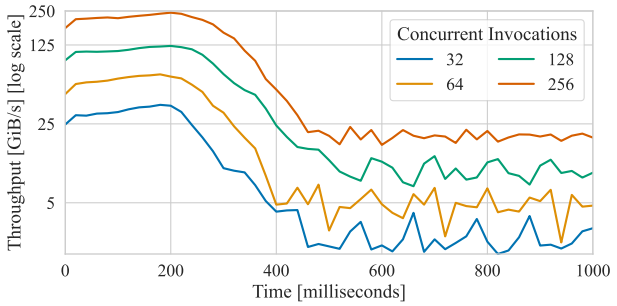

Figure 3: Aggregated network throughput for varying degrees of concurrency (32 to 256).

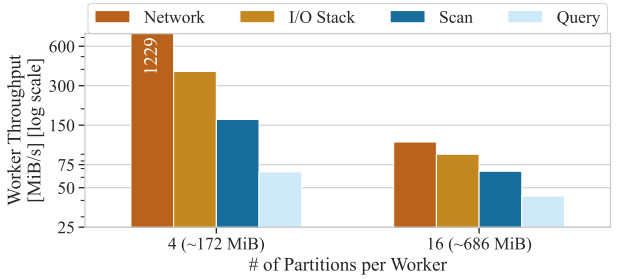

Figure 4: Query worker throughput for given input sizes within and beyond network burst budget with TPC-H Q6.

We start by running a network micro-benchmark with individual function invocations. Figure 2 shows an exemplary result. This can be used to derive the burst and baseline network throughput of cloud functions, as well as the burst capacity (of ~300 MiB here). We proceed executing the same micro-benchmark with concurrent function invocations to study the scalability of the network throughput. The engineer sees a result as in Figure 3, indicating that the throughput, in fact, scales with the number of function instances. Finally, we run a scan-heavy query, such as TPC-H Q1 (cf. query plan in Figure 7) with different input data sizes for the query workers. As shown in Figure 4, we observe that the worker throughput is significantly higher, when input sizes are smaller than the workers burst capacity, such that they can load all data with the increased burst throughput.

3.2 IOPS Scaling for Shuffle Queries

In the second scenario, we demonstrate how to exploit hot object storage with scaled out IOPS performance for queries with shuffles. Serverless query engines shuffle intermediate data through object storage. Thereby, every query worker has to potentially read all relevant columns of all assigned partitions of all workers from the preceding query stage. For large queries, shuffles may require thousands of requests and hit service quota limits of object stores, degrading query performance.

We begin with a storage micro-benchmark that employs tens to hundreds of concurrent function invocations to send small requests to the Standard and Express storage classes of S3, determining their respective IOPS performance. The results are shown in Figure 5 and indicate that S3 Express is a pre-provisioned variant of S3.

We can exploit the increased IOPS performance of S3 Express in a shuffle query like TPC-H Q12 (cf. query plan in Figure 8). We

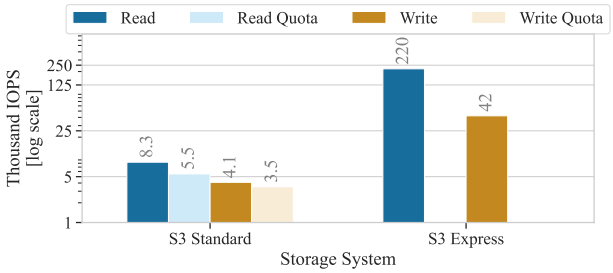

Figure 5: Operations per second and container-level quotas for the S3 storage classes Standard and Express.

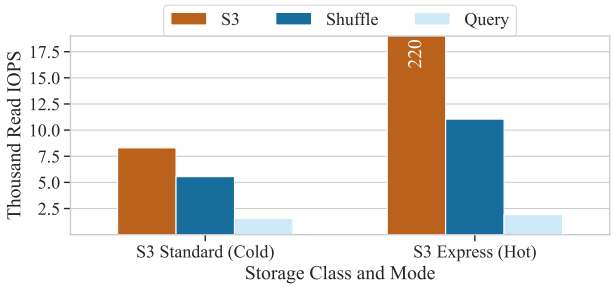

Figure 6: IOPS throughput of various S3 storage classes and their performance impact on TPC-H Q12 and its shuffle.

run the query at large scale varying the number of workers and partitions, leading to more or fewer required storage requests to execute the shuffle over S3. We see degraded performance for shuffling over S3 Standard, when the base IOPS quota is insufficient for the shuffle (see Figure 6).

Table 2: Sizing and pricing of AWS compute services.

Resource	**Lambda** (ARM) [6]	**EC2** (C6g) [2]
Memory	Configurable	Configurable
Capacity (GiB)	0.125 – 10	2 – 128
Price (¢/GiB-h)	4.80	1.70
Compute	Memory-based	Configurable
Capacity (vCPU)	0.07 – 5.79	1 – 64
Price (¢/vCPU-h)	8.49	3.40

Table 3: Performance and cost of IaaS vs. FaaS deployments.

	Query	H-Q1	H-Q12
IaaS	Runtime [s]	6.4	18.1
	Cumulated Time [s]	1266.3	5140.4
	Cost [¢]	4.79	19.58
FaaS	Runtime [s]	6.9	19.2
	Cumulated Time [s]	687.0	2,227.3
	Cost [¢]	6.49	21.19
	Storage Cost [¢]	0.16	1.39

3.3 Serverless Functions vs. Virtual Servers

In our third scenario, we compare the performance and cost of query execution on serverless vs. server-based compute instances. In either case, serverless storage is used to read base tables and

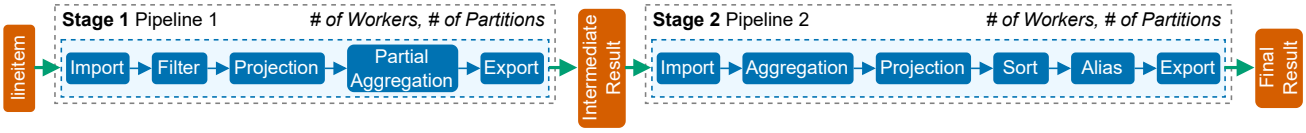

Figure 7: Query execution plan for TPC-H Q1.

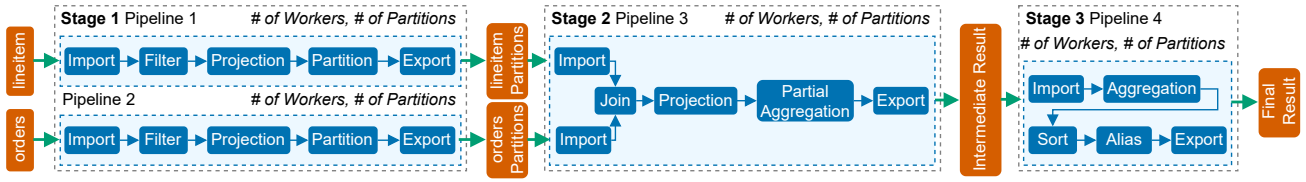

Figure 8: Query execution plan for TPC-H Q12.

shuffle intermediate results. FaaS platforms have additional performance overhead and charge higher prices per compute unit. They, however, can adapt the number of workers from query stage to stage, potentially saving resources.

We execute various queries with our query engine running on either serverless functions or (upfront started) virtual servers. For both deployment modes, we use equivalent query plans and overall physical resources in terms of vCPUs and memory.

We inspect the resulting query runtimes and cost, as shown in Table 3 with the cost based on the prices in Table 2. This allows for a discussion of the benefits of intra-query elasticity and economic viability of FaaS-based systems for infrequent vs. continuous analytical workloads.

4 CONCLUSION

We demonstrate our Skyrise framework for the evaluation of serverless cloud infrastructure. Using the framework, we offer a set of guided experiments. The experiments show characteristic features of serverless compute and storage resources. They show how these features impact the performance and economics of end-to-end data processing applications. They further allow the exploration of various parameters to develop an intuition for this new category of cloud infrastructure. Skyrise is open-source and extensible to support additional cloud services, query engines, and experiments.

ACKNOWLEDGMENTS

We would like to thank the Master's students that have worked on the Skyrise project and implemented the demonstrated components of the evaluation framework: Lars Jonas Bollmeier, David Justen, Theo Radig, and many more. This work was partially funded by SAP, the AWS Cloud Credit for Research Program, the German Research Foundation (414984028), and the European Union's Horizon 2020 research and innovation programme (957407).

REFERENCES

[1] Amazon. 2025. Amazon DynamoDB. https://aws.amazon.com/dynamodb/. Accessed: 2025-01-31.
[2] Amazon. 2025. Amazon EC2. https://aws.amazon.com/ec2/. Accessed: 2025-01-31.
[3] Amazon. 2025. Amazon EC2 C6g Instances. https://aws.amazon.com/ec2/instance-types/c6g/. Accessed: 2025-01-31.
[4] Amazon. 2025. Amazon EFS. https://aws.amazon.com/efs/. Accessed: 2025-01-31.
[5] Amazon. 2025. Amazon S3. https://aws.amazon.com/s3/. Accessed: 2025-01-31.
[6] Amazon. 2025. AWS Lambda. https://aws.amazon.com/lambda/. Accessed: 2025-01-31.
[7] Amazon. 2025. AWS Management Console. https://console.aws.amazon.com/. Accessed: 2025-01-31.
[8] Amazon. 2025. Lambda Instruction Set Architectures (ARM/x86). https://docs.aws.amazon.com/lambda/latest/dg/foundation-arch.html. Accessed: 2025-01-31.
[9] Amazon. 2025. Serverless on AWS. https://aws.amazon.com/serverless/. Accessed: 2025-01-31.
[10] Michael Armbrust, Armando Fox, Rean Griffith, Anthony D. Joseph, Randy H. Katz, Andy Konwinski, Gunho Lee, David A. Patterson, Ariel S. Rabkin, Ion Stoica, and Matei A. Zaharia. 2009. *Above the Clouds : A Berkeley View of Cloud Computing.* Technical Report UCB/EECS-2009-28. EECS Department, University of California, Berkeley.
[11] Thomas Bodner. 2020. Elastic Query Processing on Function as a Service Platforms. In *VLDB PhD Workshop.*
[12] Thomas Bodner, Tobias Pietz, Lars Jonas Bollmeier, and Daniel Ritter. 2022. Doppler: Understanding Serverless Query Execution. In *SIGMOD BiDEDE.* ACM, 2:1–2:4.
[13] Thomas Bodner, Theo Radig, David Justen, Daniel Ritter, and Tilmann Rabl. 2025. An Empirical Evaluation of Serverless Cloud Infrastructure for Large-Scale Data Processing. *arXiv preprint arXiv:2501.07771* (2025).
[14] Thomas Bodner, Daniel Ritter, Martin Boissier, and Tilmann Rabl. 2025. Skyrise: Exploiting Serverless Cloud Infrastructure for Elastic Data Processing. *arXiv preprint arXiv:2501.08479* (2025).
[15] Benoît Dageville, Thierry Cruanes, Marcin Zukowski, Vadim Antonov, Artin Avanes, Jon Bock, Jonathan Claybaugh, Daniel Engovatov, Martin Hentschel, Jiansheng Huang, Allison W. Lee, Ashish Motivala, Abdul Q. Munir, Steven Pelley, Peter Povinec, Greg Rahn, Spyridon Triantafyllis, and Philipp Unterbrunner. 2016. The Snowflake Elastic Data Warehouse. In *SIGMOD.* ACM, 215–226.
[16] Eric Jonas, Johann Schleier-Smith, Vikram Sreekanti, Chia-che Tsai, Anurag Khandelwal, Qifan Pu, Vaishaal Shankar, João Carreira, Karl Krauth, Neeraja Jayant Yadwadkar, Joseph E. Gonzalez, Raluca Ada Popa, Ion Stoica, and David A. Patterson. 2019. *Cloud Programming Simplified: A Berkeley View on Serverless Computing.* Technical Report UCB/EECS-2019-3. EECS Department, University of California, Berkeley.
[17] Jupyter contributors. 2025. Project Jupyter: Free Software, Open Standards, and Web Services for Interactive Computing Across All Programming Languages. https://jupyter.org/. Accessed: 2025-01-31.
[18] Sergey Melnik, Andrey Gubarev, Jing Jing Long, Geoffrey Romer, Shiva Shivakumar, Matt Tolton, Theo Vassilakis, Hossein Ahmadi, Dan Delorey, Slava Min, Mosha Pasumansky, and Jeff Shute. 2020. Dremel: A Decade of Interactive SQL Analysis at Web Scale. *PVLDB* 13, 12 (2020), 3461–3472.
[19] Ingo Müller, Renato Marroquín, and Gustavo Alonso. 2020. Lambada: Interactive Data Analytics on Cold Data Using Serverless Cloud Infrastructure. In *SIGMOD.* ACM, 115–130.
[20] Qifan Pu, Shivaram Venkataraman, and Ion Stoica. 2019. Shuffling, Fast and Slow: Scalable Analytics on Serverless Infrastructure. In *NSDI.* USENIX, 193–206.
[21] Joel Scheuner and Philipp Leitner. 2020. Function-as-a-Service Performance Evaluation - A Multivocal Literature Review. *Journal of Systems and Software* 170 (2020), 110708.
[22] Transaction Processing Performance Council. 2025. Specification of the TPC-H Benchmark. https://www.tpc.org/tpch/. Accessed: 2025-01-31.
[23] Transaction Processing Performance Council. 2025. Specification of the TPCx-BB Benchmark. https://www.tpc.org/tpcx-bb/. Accessed: 2025-01-31.
[24] Midhul Vuppalapati, Justin Miron, Rachit Agarwal, Dan Truong, Ashish Motivala, and Thierry Cruanes. 2020. Building An Elastic Query Engine on Disaggregated Storage. In *NSDI.* USENIX, 449–462.