Serverful Functions: Leveraging Servers in Complex Serverless Workflows (industry track)

Germán T. Eizaguirre
germantelmo.eizaguirre@urv.cat
Universitat Rovira i Virgili
Tarragona, Spain

Daniel Barcelona-Pons
daniel.barcelona@urv.cat
Universitat Rovira i Virgili
Tarragona, Spain

Aitor Arjona
aitor.arjona@urv.cat
Universitat Rovira i Virgili
Tarragona, Spain

Gil Vernik*
gilv@il.ibm.com
IBM Research
Haifa, Israel

Pedro García-López
pedro.garcia@urv.cat
Universitat Rovira i Virgili
Tarragona, Spain

Theodore Alexandrov†
theodore.alexandrov@embl.de
EMBL
Heidelberg, Germany

Abstract

The scalability of cloud functions makes them a convenient backend for elastic data analytics pipelines where parallelism changes drastically from one stage to the next. However, cloud functions require intermediate storage systems for communication, which limits the efficiency of stateful operations. Furthermore, cloud functions are expensive, which reduces the cost-effectiveness of pure serverless architectures. We propose a hybrid architecture for data analytics that uses cloud functions for embarrassingly parallel stages and virtual cloud instances for stateful operations under a unified serverless programming framework. Extending Lithops, a serverless programming library, we implement a parallel programming interface that proactively provisions serverless and serverful cloud resources with minimal user intervention. We validate the feasibility of a hybrid architecture, by comparing it to fully serverless and serverful versions of a production-level metabolomics pipeline. We show that mixing cloud functions with virtual instances increases the cost-effectiveness of the execution by up to 188.23% over the serverless implementation, while achieving a speedup of 3.64 compared to the serverful one.

CCS Concepts

• **Computer systems organization** → **Cloud computing**; • **Theory of computation** → *Distributed computing models.*

Keywords

Cloud computing, serverless computing, function-as-a-service, resource allocation, resource efficiency

ACM Reference Format:
Germán T. Eizaguirre, Daniel Barcelona-Pons, Aitor Arjona, Gil Vernik, Pedro García-López, and Theodore Alexandrov. 2024. Serverful Functions: Leveraging Servers in Complex Serverless Workflows (industry track). In *25th International Middleware Conference Industrial Track (MIDDLEWARE Industrial Track '24), December 2–6, 2024, Hong Kong, Hong Kong*. ACM, New York, NY, USA, 7 pages. https://doi.org/10.1145/3700824.3701095

*This work was done while this author was affiliated with IBM Research.
†This work was done while this author was affiliated with EMBL.

1 Introduction

Serverless services are a practical approach to making the Cloud more accessible, as much of the management burden is abstracted away by the cloud provider. Serverless functions —or Function as a Service (*FaaS*)— such as AWS Lambda [3] are the leading embodiment of the serverless paradigm: stateless, event-driven, ephemeral computations with low start-up latency and broad concurrency. FaaS provides a convenient pay-as-you-go billing model and intuitive, easy-to-use abstractions over the underlying resources. Developers can deploy their code to the Cloud in seconds, running in a secure, isolated and managed environment that auto-scales during peak demand and to zero when there is none.

The potential of serverless services has not gone unnoticed in data analytics; particularly in elastic multi-stage workloads where the number of parallel tasks changes drastically from one stage to another. Traditional cluster technologies on virtual machines (VMs) often fall short in such scenarios due to expecting a fixed pool of resources or slow (if any) auto-scaling capabilities. These conditions lead to a mismatch between cluster size and resource requirements in elastic multi-stage workloads, resulting in overprovisioning (where more resources are provisioned than needed, leading to inflated costs), or underprovisioning (where fewer resources are provisioned than required, leading to degraded performance). Serverless functions address this issue more efficiently, quickly scaling out and in based on the needs of the workload, and fitting the exact resource requirements of each stage.

However, serverless functions face a major limitation: they are ephemeral, making them non-addressable and stateless, thus dependent on external disaggregated storage for communication. Functions are not suitable for complex communication patterns like sorting, grouping, or re-partitioning, which require extensive data sharing between workflow stages. These *stateful operations* are a significant performance challenge for serverless data analytics due to the load placed on the storage. Overloading the storage slows down I/O throughput, reduces CPU utilisation, and hurts cost-efficiency. As the number of functions per stage increases, so does the number of storage requests, gradually degrading I/O performance and exacerbating resource waste.

Service	Execution time
AWS Lambda	12.56 s
AWS EC2	42.34 s
AWS EMR Serverless	134.87 s

Table 1: Total execution time of a `map` operation involving 100 CPU-bound functions, each with a runtime of five seconds, measured across multiple services. Times include resource provisioning and deprovisioning time. AWS Lambda uses 1769 MB of memory per function. EC2 uses an m6a.32xlarge instance with 128 vCPUs created from a pre-built AMI. EMR Serverless runs with its default execution parameters.

The drawbacks of serverless functions motivate the exploration of alternatives for extensive data sharing between stages. We argue that, with sufficient abstraction, serverful (VMs) and serverless (functions) resources can be seamlessly combined within a complex workflow. By deploying servers in a *serverless* flavour, i.e., launching select workflow stages on a short-lived, right-sized VM, we mitigate the limitations of functions to achieve fast state sharing in stateful phases with minimal developer intervention. With proper optimisation, the overhead of creating VMs becomes feasible to launch stateful stages (Table 2), as the cost of (indirect) communication in serverless functions would be prohibitive.

Combining the virtues of serverful and serverless components in the same workload has been hinted previously [15, 24, 28], but its viability has not been proven in real environments. In this article, we bridge the gap between serverless and serverful technologies by transparently integrating VMs into serverless analytics. We do this by extending Lithops, a serverless programming library, with the ability to leverage hybrid architectures. With minimal in-code changes, a developer can choose to seamlessly run their parallel code on either serverless functions or VMs.

Hybrid serverless architectures: what fits?

Hybrid architectures are well-suited for big data pipelines that combine stateful data preparation with embarrassingly parallel data processing stages— a common pattern in fields such as omics [5, 19]. By integrating cloud VMs and cloud functions, we enable rapid vertical scalability, which is crucial for in-place data preparation, alongside rapid horizontal scalability to efficiently manage embarrassingly parallel stages.

We demonstrate the potential of hybrid architectures by optimising a production-ready multi-stage metabolomics pipeline. Developers at the European Molecular Biology Laboratory (EMBL) employ an image spectrometry annotation pipeline that originally run on a fixed, serverful Spark deployment in the Cloud that was difficult to scale and provision at the right size. To manage the high elasticity of the pipeline, they chose to leverage *FaaS* as their computation backend [36]. However, the initial implementation suffered from the limitations of cloud functions for stateful operations, especially as the input size increased and the number of parallel functions scaled out. With the contributions of this paper, the pipeline uses a novel architecture that successfully combines cloud functions with VMs, vertically scaled on demand for stateful operations. Overall, the hybrid deployment of the pipeline is 188.23% more cost-efficient than a deployment based entirely on cloud functions, while still achieving a 3.64× speedup over the original serverful architecture.

2 Serverful backends for serverless analytics

2.1 Integration of serverful backends

We construct our framework for hybrid pipeline development on Lithops, an out-of-the-box Python framework for serverless data analytics. Function executors are the basic handlers in Lithops: through a set of straightforward primitives, executors port parallel function calls to cloud functions, keeping developers agnostic about background resource management. The original architecture of Lithops executors launches one logical function per cloud function instance and automatically monitors its execution via object storage. We extend Lithops with a set of serverful backends for different cloud providers, allowing users to launch logical functions in cloud virtual server instances (VMs). We automate the deployment and configuration of VMs using custom images, including their complete set of dependencies. It is the executor that chooses, based on user-defined preference, to use existing, previously configured VMs, or to create new ones.

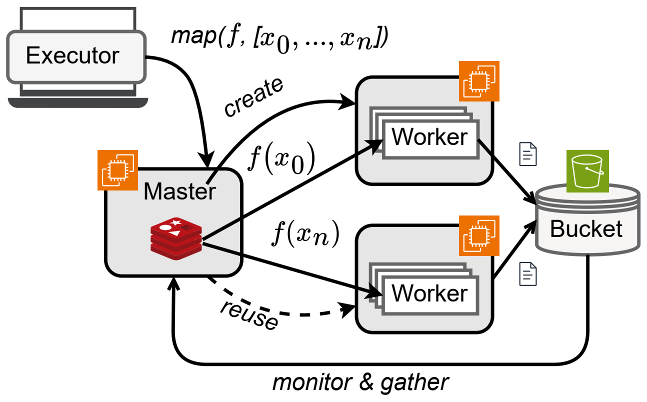

Figure 1: Serverless functions over serverful resources with Lithops on AWS.

Figure 1 illustrates how we perform serverless function calls over serverful virtual instances. For each concurrent call, the executor connects to a master instance on a dedicated VM that is provisioned and managed by the Lithops client via SSH. The master identifies available resources and proactively provisions the required VMs for the call, if necessary. It then creates workers in the VMs via SSH, based on their CPU capacity. Workers load logical functions from a Redis server running in the master, execute them and write their results back to object storage, while the master monitors function completion and collects the output. Once all logical functions have been completed, all resources are automatically stopped.

Overall, serverful execution is performed in a serverless manner, automatically provisioning and deprovisioning resources and keeping virtual instances active only for the execution time. Given that cloud functions are inherently ephemeral and stateless, each of its logical functions operates in a separate Docker container. Workers within a VM, instead, run as processes within the same container, enabling logical functions to access the same shared memory namespace and exploit locality in data exchanges.

```
from lithops import FunctionExecutor, Storage

storage = Storage()

def create(x):
    return storage.put_cloudobject(x)

def double(cobj):
    obj = storage.get_cloudobject(cobj)
    obj *= 2
    return storage.put_cloudobject(obj)

# Lambda execution
exec = FunctionExecutor(backend = "aws_lambda")
exec.map(create, ["a", "b", "c"])
cobjs = fexec.get_result()

# VM execution
exec = FunctionExecutor(backend = "aws_ec2")
exec.map(double, cobjs)
cobjs = fexec.get_result()

objs = [storage.get_cloudobject(o) for o in cobjs]
print(objs)
> ["aa", "bb", "cc"]
```

Listing 1: Execution of parallel code with Lithops hybrid backends.

2.2 Lithops' seamless programming framework

Our objective with Lithops is to optimize resource transparency in Cloud development. With that in mind, we minimize code changes to use one backend or another (serverless or serverful). The entry point to Lithops execution is the `FunctionExecutor`, which deploys the scheduler and conducts the subsequent function calls. As all executors share the same primitives, users only need to specify their desired backend and run their code. Stages running on different backends can easily communicate data using Lithops' native `CloudObjects`. `CloudObjects` run over the underlying object storage system and provide an intuitive interface for sharing Python objects between decoupled components. Taken together, the data sharing abstractions and the serverless and serverful function executors provide an out-of-the-box interface to seamlessly implement hybrid architectures, as exemplified in Listing 1.

Lithops is the result of the joint efforts of Universitat Rovira i Virgili (URV) and IBM, integrating the refined outcomes of the CloudButton[1], Neardata[2] and CloudSkin[3] European projects, and it has been developed incrementally [31] to its ongoing stable version. The backends for serverful resources proposed in the present work are publicly available in Lithops' Github repository: https://github.com/lithops-cloud/lithops.

3 Related work

Research in serverless analytics is prosperous, specifically in the use of cloud functions for data analytics. PyWren [21] proposes a programmatic framework for embarrassingly parallel jobs over AWS Lambda, while mu [14] and gg [13] port local parallel applications to cloud functions using lightweight containers. Wukong [7] executes directed acyclic graphs (DAGs) over cloud functions with a distributed recursive scheduling algorithm. There are even proposals for running local multiprocessing libraries transparently on top of cloud functions [4]. However, none of these allow explicit switching between cloud functions and alternative cloud backends via the same programming framework.

State sharing between cloud functions has also been covered previously [6, 22, 35], but mostly focused on non data-intensive coordination. The literature on optimising data exchanges across cloud object storage is broad [26, 32, 33]. However, they share the problem of throughput saturation at high levels of parallelism. Locus [28] introduces the idea of augmenting object storage with Redis instances [30], but delegating its provisioning and management to the user. We instead focus on delivering a completely serverless experience to the end developer.

The use of virtual instances for serverless exchanges is addressed by SPRIGHT [29], which demonstrates the benefits of passing data through shared memory in serverless environments. SONIC [24] identifies the optimal method for communicating functions based on the workload and places functions accordingly. They evaluate the use of intermediate virtual machines, object storage and shared memory. However, neither SPRIGHT nor SONIC can run in commercial cloud functions and require a dedicated cloud deployment. Lithops functions fully work over commercial cloud services.

Hybrid architectures stand out in the data analytics ecosystem for their agility and flexibility. Some established frameworks, such as Dryad [20], directly do not have out-of-the-box autoscaling capabilities. Others do provide resource autoscaling —e.g., Nextflow [11], Dask [10] or cloud-hosted Spark services [1, 2, 17, 18]. However, their autoscaling is reactive, slowly adjusting resources based on real-time workload demands, which is not a match for the rapid deployment of cloud functions. Additionally, they typically scale only horizontally, making fully in-memory operations unfeasible for large input sizes. BigQuery [16], a fully managed database service, provides automatic and high performance scalability but is focused on SQL queries, rather than domain-specific pipeline programming, and it can result in higher billing than cloud functions at stateful operations [32].

Lithops, along with Modal [25] and Coiled [8], is the only out-of-the-box, actively maintained serverless data analytics framework. However, Modal and Coiled charge users supplementary commissions for managing cloud resources. Lithops is open source, does not incur into additional fees and runs directly on the developer's cloud account, so developers have complete transparency over their workloads and only pay for their Cloud utilisation. It has been and is being used as an architectural building block in genomics [5], geospatial analysis [9] and machine learning [34]. For example, the European Molecular Biology Laboratory's (EMBL) METASPACE [27] is currently using Lithops at production level in its metabolomics data space. The number of cloud computing research prototypes based on Lithops is also broad [6, 12, 23, 32, 33]. Its maintainers have collected a representative set of serverless

[1]https://cloudbutton.eu/
[2]https://neardata.eu/
[3]https://cloudskin.eu/

pipelines that validate its adaptability, the compilation of which is publicly available.[4]

4 Use case validation

4.1 Serverless data analytics in a metabolomics data space

To illustrate the value of integrating serverful components into serverless pipelines, we will describe our experience optimising the METASPACE metabolomics annotation pipeline. Within the METASPACE data hub [27], users are provided an annotation pipeline to identify metabolites in imaging mass spectrometry ((I)MS) datasets —i.e., ion spectra measurements of narrow biological tissue sections. The annotation pipeline receives as input (1) a dataset, the result of an IMS essay and (2) a database of formulas with predictable MS signals. The signals ascribed to the pixels of the dataset —each corresponding to a physical location in the tissue— are compared with the signals in the database. As a result, plausible metabolites in the sample are detected, along with their location. The pipeline is currently publicly available and actively used by hundreds of researchers worldwide.

Annotation starts with database formula generation, with a maximum of a few hundred parallel tasks. It then sorts and partitions the input dataset and database over a series of stateful operations. Finally, the dataset and database are compared in a series of embarrassingly parallel stages. With dataset sizes ranging from a few MBs to hundreds of GBs, the maximum parallelism of the pipeline can vary significantly between workloads. In fact, metabolite annotation shows great elasticity, as the parallelism of a workload ranges from modestly parallel stages (e.g., 32 tasks in database partitioning) to massive concurrency in its final stages (reaching up to a few thousand parallel tasks in a typical annotation job). The increase in parallelism is super-linear with respect to the size of the dataset, as the comparison between the dataset and the database formulas is Cartesian, requiring a data-driven, scalable architecture.

METASPACE has successfully migrated from a cluster-based Spark implementation of this pipeline to a Lithops implementation using cloud functions. Metabolite annotation is an ideal candidate for serverless architectures: with data volumes ranging from a few to hundreds of gigabytes and the inherent elasticity of workloads, it is impractical to allocate precise resources for every job's needs. Cloud functions allow us to provision the exact resources required for each stage, as illustrated in Figure 2, quickly and efficiently.

We compare the Spark implementation of the pipeline, using the original configuration adopted by METASPACE for its data hub —four c5.4x large AWS EC2 instances, providing a total of 64 vCPUs and 128GB of physical memory— with a serverless implementation on Lithops, using AWS Lambda and AWS S3 as the compute and storage backends, respectively. We exclude cluster configuration and initialisation times from our experiments, and run them on an out-of-the-box deployment. As annotation jobs vary in terms of imaging dataset and molecular database size, we evaluate three different jobs, which are roughly characterised in Table 2. *Brain* is a small, testbed input, while *Xenograft* would have the closest traits to a typical METASPACE job.

[4]https://github.com/iAmJK44/serverless_benchmarks

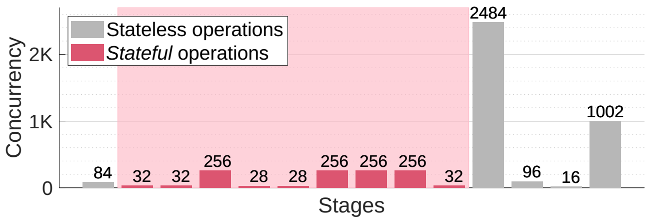

Figure 2: Number of concurrent cloud functions per stage in the serverless annotation of *Xenograft*. Stages involved in *stateful* operations are represented in red, while fully stateless stages are grey.

Figure 3 shows the execution time of the jobs studied in the serverless implementation and in Spark. Using serverless components instead of a cluster with a fixed pool of VMs can reduce annotation time by up to 81%. As input data increases, the number of resources required in terms of vCPUs and memory becomes more pressing, eventually leading to cluster under-provisioning. Even when provisioned at runtime, the elasticity of serverless resources compensates for their startup time in typical annotation jobs, with a speedup of 2.50 in *Xenograft*.

In Table 3 we examine the resource utilisation of each implementation by measuring its CPU usage during *Xenograft*. In the Spark implementation, the available compute resources remain constant from end to end. With serverless components, however, resources depend on the number of functions provisioned and running at a given time —including the Lithops scheduler. As expected, the serverless architecture delivers an average of 20% better CPU utilisation, as functions are provisioned and deprovisioned elastically. This stabilises resource utilisation, reducing the standard deviation by more than 20%. Similarly, the absolute minimum CPU usage with cloud functions is 35%, far from 0%, as resources are rapidly deprovisioned after each stage, effectively mitigating possible underprovisioning scenarios.

4.2 The serverless sort hindrance

Overall CPU usage masks the problem of data sharing, though. Segregating stateful operations at Table 3 discloses a resource utilisation unefficiency in both implementations, as CPU usage does not surpass 50% in none of them. In the serverless deployment, a prolonged underutilisation of CPU resources is associated with intensive communication and its consequent increase of I/O latency. This is the case for the distributed dataset and database sorting and partitioning operations, which require I/O-intensive all-to-all exchanges through AWS S3 [33] and comprise several stages in the pipeline.

Underutilising serverless resources comes at a price. Considering an on-demand c5x4large EC2 instance, the cost of a vCPU is 0.12e−4 \$/s.[5] AWS Lambda, on the other hand, given that 1769 MB of run-time memory is the equivalent of a single vCPU,[6] is charged at 0.28e−4 \$/s[7] (both services running at `us-east`, 30 June 2024). As well as doubling the cost of VMs per second, AWS Lambda also

[5]https://aws.amazon.com/ec2/pricing/on-demand/
[6]https://docs.aws.amazon.com/lambda/latest/dg/configuration-memory.html
[7]https://aws.amazon.com/lambda/pricing

Dataset name	Abbreviation	Dataset size (GB)	Database size (#formulas)	Max Data Volume (GB)
Brain02_Bregma1-42_02	*Brain*	0.05	12k	37.45
CT26_xenograft	*Xenograft*	1.80	74k	235.98
X089-Mousebrain_842x603	*X089*	7.01	29k	174.33

Table 2: Proposed METASPACE job setups. "Dataset size" represents the size of the imaging spectrometry sample. "Database size" counts the number metabolites, whose spectra and its derivatives' will be inspected in the dataset. Last column shows the maximum data volume that is processed in a single stage.

CPU Usage	Cloud functions	Spark
Average	72.76%	53.53%
Standard deviation	19.02%	42.19%
Maximum	99.99%	99.43%
Minimum	35.58%	0.43%
Average (*stateful* operations)	**40.57%**	**17.68%**

Table 3: CPU usage of the *Xenograft* annotation running on cloud functions compared to a production-matched Spark cluster.

allocates a greater proportion of resources to massively parallel stages. Thus, wasting resources in cloud functions comes at a much bigger price than in virtual instances.

Figure 4 shows that an architecture based entirely on cloud functions is more expensive than a cluster, despite being significantly more performant. The cost of serverless executions doubles that of clusters for typical annotation jobs, and it is up to nearly 4x more expensive for demanding jobs. Overall, cloud functions meet the needs of elasticity, but fail in stateful stages and can increase the cost of the pipeline when strictly adhering to them.

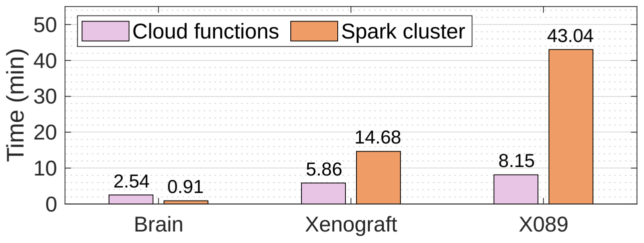

Figure 3: Execution time of different METASPACE annotation jobs, run on Lithops over cloud functions against a production-matched Spark cluster.

To prove that cost can be mitigated by better management of stateful operations, we study the case of sorting and partitioning an input dataset. We process *Xenograft* using two different architectures. On the one hand, a distributed sort purely based on cloud functions and object storage, as we do in the serverless deployment of the pipeline. On the other hand, an in-place sort in a VM, sharing data in local memory instead of object storage. We use a total of 64 GB of physical memory in both. For the serverless deployment, we use 37 AWS Lambda functions, each with 1769 MB of memory.

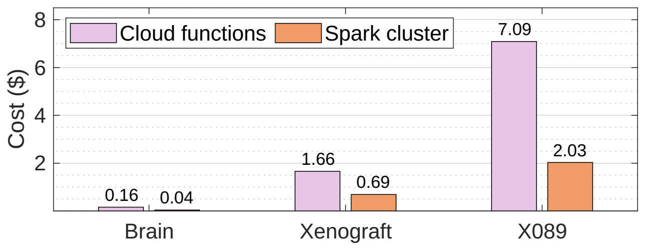

Figure 4: Cost, in dollars, of distinct METASPACE annotation jobs, run on Lithops over cloud functions against a production-matched Spark cluster.

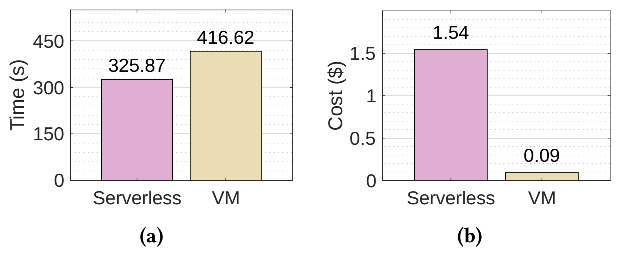

Figure 5: Distributed sort results for *Xenograft* on a cloud function-based serverless implementation and on a single VM. (a) Total execution time. (b) Total cost.

Since 1769 MB provides the equivalent of a single vCPU in AWS Lambda, the serverless deployment employs a total of 37 vCPUs. In the VM implementation, we use an m4.4xlarge AWS EC2 instance with 16 vCPUs and 64 GB of RAM.

Figure 5 shows the execution time and cost results for both implementations. Although the serverless implementation provides a 1.28 speedup over the VM implementation —which includes VM initialisation time from a previously built AMI— the cost differences are significant: sorting and partitioning in the VM is 17 times cheaper than using cloud functions. Provisioning the same amount of memory and even doubling the number of vCPUs, the speedup of serverless is opaqued by its cost implications.

Resource starvation is again a significant contributor to cost inefficiency. Both implementations spend a large proportion of their execution time on I/O, 48.89% in the serverless execution and 53.24% in the VM execution, during which CPU resources are idle. However, the asymmetry in cost per vCPU makes the serverless I/O much more expensive than the serverful. The time for reading, exchanging and writing data with cloud functions is charged at \$0.75, while with a VM it costs only \$0.05.

Our analysis reinforces the idea of using VMs for stateful operations. One of the likely advantages of cloud functions over serverful services could be their scalability, even to thousands of concurrent functions. With a few GBs of memory each, a pool of cloud functions can handle hundreds of gigabytes of input. For example, AWS Lambda can run tens of thousands of concurrent functions with 10 GB of memory each.[8] However, the range of virtual instance specifications in cloud provider catalogues is also extensive. AWS EC2, for example, offers instances with tens of terabytes of memory,[9] which exceeds even the aggregated memory of cloud functions. We could virtually sort datasets of thousands of GBs within serverful components, vertically scaling them to input size.

4.3 Convenient integration of serverless and serverful components

We adhere to our serverful component integration bid and the bilateral analysis of the sort operation to optimize the annotation pipeline. Embarrassingly parallel stages, such as data processing, benefit most from *FaaS*, while serverful services are better suited to stateful stages with strong dependencies. Exploiting Lithops's unified programming framework, we develop a hybrid serverless architecture selectively choosing the right service for each stage. We use AWS Lambda for embarrassingly parallel stages in Figure 2, while hosting stateful operations in properly scaled EC2 instances.

Accurately defining the memory requirements for each input is a non-trivial challenge, as sorting is a memory-intensive operation that consumes up to 2-3 times the data size. Our architecture measures input size and selects the host instance type based on empirically defined bounds. Data is partitioned, processed as numpy arrays and converted to pandas partitions that are written back to object storage. Reads and writes are parallelized to optimise communication and to overlap (de)serilization with I/O.

To evaluate the deployment that makes the most efficient use of resources, we integrate latency and cost into a single metric, cost-performance, defined as $\frac{1}{latency \times cost}$. An architecture that maximises cost-performance is considered to deliver better performance relative to its cost. By running sorts and partitions in a single VM, we improved cost-performance by 188.23% in *Xenograft* and 148.10% in *X089* compared to the serverless implementation. Figure 6 shows that the hybrid implementation improves the cost performance of the cloud functions-based implementation in all jobs.

The end-to-end execution times of the studied deployments are listed in Table 4. When running the annotation pipeline in a hybrid architecture, we achieve a speedup of 3.64 in *X089* and 2.21 in *Xenograft* compared to Spark. Overall, by coherently alternating cloud functions and VMs, we achieve up to 75% better performance at a similar cost to a cluster implementation through a smarter use of resources.

[8]https://docs.aws.amazon.com/lambda/latest/dg/gettingstarted-limits.html
[9]https://aws.amazon.com/es/ec2/instance-types/u7i

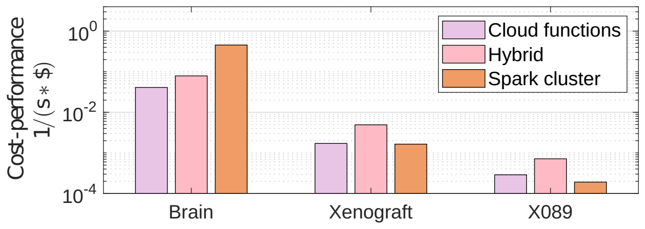

Figure 6: Cost-performance (in $\frac{1}{latency \times cost}$) of distinct METASPACE annotation jobs in the three studied architectures.

Job	Cloud functions	Hybrid	Spark
Brain	152.20s	105.49s	54.83s
Xenograft	351.57s	398.70s	889.54s
X089	488.86s	709.14s	2582.66s

Table 4: Execution time of each annotation job executed in the studied architectures.

The proposed hybrid architecture has been adopted by EMBL and is the default deployment of the annotation pipeline in the METASPACE dataspace.[10] Lithops's programming abstractions made it easy to integrate VMs into the original serverless architecture.

5 Conclusion

Because of their scalability, cloud functions adapt to the needs of parallel, multi-stage jobs and avoid under and overutilisation of resources, resulting in cost and performance improvements. However, their decoupled nature makes them suboptimal for stateful operations. In this paper, we present the idea of seamlessly integrating VMs into stateful stages of serverless pipelines through appropriate resource abstractions. We use Lithops, a serverless analytics framework, and extend it with a VM management functionality. To validate our proposal, we optimise the METASPACE metabolite annotation pipeline, from a pure serverless implementation to a hybrid serverless-serverful deployment. We showcase the benefits of integrating serverful components into the serverless paradigm keeping the execution of embarrassingly parallel stages in cloud functions while running I/O-intensive stages in VMs. By using virtual instances in the Cloud, we were able to significantly reduce the execution time of the pipeline while keeping its overall cost unchanged.

Acknowledgments

This work has been partly funded by the EU Horizon programme under the grant agreements: 101092644 (NearData), 101092646 (CloudSkin), and 101093110 (EXTRACT). Germán T. Eizaguirre is recipient of a pre-doctoral FPU grant from the Spanish Ministry of Universities (ref. FPU21/00630). Special thanks to Pau Balanzà-Malagelada and Jordi Canosa-Casellas for their support in the experimentation phase.

[10]https://github.com/metaspace2020/metaspace/tree/master/metaspace/engine

References

[1] Amazon. 2024. AWS EMR. https://aws.amazon.com/emr

[2] Amazon. 2024. AWS EMR Serverless. https://aws.amazon.com/emr/serverless/

[3] Amazon. 2024. AWS Lambda. https://aws.amazon.com/lambda/

[4] Aitor Arjona, Gerard Finol, and Pedro García López. 2023. Transparent serverless execution of Python multiprocessing applications. *Future Generation Computer Systems* 140 (2023), 436–449. https://doi.org/10.1016/j.future.2022.10.038

[5] Aitor Arjona, Arnau Gabriel-Atienza, Sara Lanuza-Orna, Xavier Roca-Canals, Ayman Bourramouss, Tyler K. Chafin, Lucio Marcello, Paolo Ribeca, and Pedro García-López. 2023. Scaling a Variant Calling Genomics Pipeline with FaaS. In *Proceedings of the 9th International Workshop on Serverless Computing (WoSC '23)*. Association for Computing Machinery, New York, NY, USA, 59–64. https://doi.org/10.1145/3631295.3631403

[6] Daniel Barcelona-Pons, Pierre Sutra, Marc Sánchez-Artigas, Gerard París, and Pedro García-López. 2022. Stateful Serverless Computing with Crucial. *ACM Trans. Softw. Eng. Methodol.* 31, 3, Article 39 (2022), 38 pages. https://doi.org/10.1145/3490386

[7] Benjamin Carver, Jingyuan Zhang, Ao Wang, Ali Anwar, Panruo Wu, and Yue Cheng. 2020. Wukong: a scalable and locality-enhanced framework for serverless parallel computing. In *Proceedings of the 11th ACM Symposium on Cloud Computing* (Virtual Event, USA) *(SoCC '20)*. Association for Computing Machinery, New York, NY, USA, 1–15. https://doi.org/10.1145/3419111.3421286

[8] Coiled. 2024. Coiled. https://www.coiled.io/

[9] Cubed. 2024. Cubed: Bounded-memory serverless array processing in xarray. https://xarray.dev/blog/cubed-xarray

[10] Dask Development Team 2016. *Dask: Library for dynamic task scheduling*. Dask Development Team. http://dask.pydata.org

[11] Paolo Di Tommaso, Maria Chatzou, Evan W Floden, Pablo Prieto Barja, Emilio Palumbo, and Cedric Notredame. 2017. Nextflow enables reproducible computational workflows. *Nature biotechnology* 35, 4 (2017), 316–319. https://doi.org/10.1038/nbt.3820

[12] Gerard Finol, Gerard París, Pedro García-López, and Marc Sánchez-Artigas. 2024. Exploiting inherent elasticity of serverless in algorithms with unbalanced and irregular workloads. *J. Parallel Distrib. Comput.* 190, C (2024), 15 pages. https://doi.org/10.1016/j.jpdc.2024.104891

[13] Sadjad Fouladi, Francisco Romero, Dan Iter, Qian Li, Shuvo Chatterjee, Christos Kozyrakis, Matei Zaharia, and Keith Winstein. 2019. From laptop to lambda: outsourcing everyday jobs to thousands of transient functional containers. In *Proceedings of the 2019 USENIX Annual Technical Conference* (Renton, WA, USA) *(USENIX ATC '19)*. USENIX Association, USA, 475–488.

[14] Sadjad Fouladi, Riad S. Wahby, Brennan Shacklett, Karthikeyan Vasuki Balasubramaniam, William Zeng, Rahul Bhalerao, Anirudh Sivaraman, George Porter, and Keith Winstein. 2017. Encoding, fast and slow: low-latency video processing using thousands of tiny threads. In *Proceedings of the 14th USENIX Conference on Networked Systems Design and Implementation* (Boston, MA, USA) *(NSDI'17)*. USENIX Association, USA, 363–376.

[15] Pedro García-López, Marc Sánchez-Artigas, Simon Shillaker, Peter Pietzuch, David Breitgand, Gil Vernik, Pierre Sutra, Tristan Tarrant, and Ana Juan Ferrer. 2019. ServerMix: Tradeoffs and Challenges of Serverless Data Analytics. arXiv:1907.11465

[16] Google. 2024. GCP BigQuery. https://cloud.google.com/bigquery?

[17] Google. 2024. GCP Dataproc. https://cloud.google.com/dataproc

[18] Google. 2024. GCP Dataproc Serverless. https://cloud.google.com/dataproc-serverless/docs

[19] Piotr Grzesik, Dariusz R Augustyn, Łukasz Wyciślik, and Dariusz Mrozek. 2021. Serverless computing in omics data analysis and integration. *Briefings in Bioinformatics* 23, 1 (09 2021). https://doi.org/10.1093/bib/bbab349

[20] Michael Isard, Mihai Budiu, Yuan Yu, Andrew Birrell, and Dennis Fetterly. 2007. Dryad: distributed data-parallel programs from sequential building blocks. In *Proceedings of the 2nd ACM SIGOPS/EuroSys European Conference on Computer Systems 2007* (Lisbon, Portugal) *(EuroSys '07)*. Association for Computing Machinery, New York, NY, USA, 59–72. https://doi.org/10.1145/1272996.1273005

[21] Eric Jonas, Qifan Pu, Shivaram Venkataraman, Ion Stoica, and Benjamin Recht. 2017. Occupy the cloud: distributed computing for the 99%. In *Proceedings of the 2017 Symposium on Cloud Computing* (Santa Clara, CA, USA) *(SoCC '17)*. Association for Computing Machinery, New York, NY, USA, 445–451. https://doi.org/10.1145/3127479.3128601

[22] Anurag Khandelwal, Yupeng Tang, Rachit Agarwal, Aditya Akella, and Ion Stoica. 2022. Jiffy: elastic far-memory for stateful serverless analytics. In *Proceedings of the Seventeenth European Conference on Computer Systems* (Rennes, France) *(EuroSys '22)*. Association for Computing Machinery, New York, NY, USA, 697–713. https://doi.org/10.1145/3492321.3527539

[23] Pedro García López, Aitor Arjona, Josep Sampé, Aleksander Slominski, and Lionel Villard. 2020. Triggerflow: trigger-based orchestration of serverless workflows. In *Proceedings of the 14th ACM International Conference on Distributed and Event-Based Systems* (Montreal, QC, Canada) *(DEBS '20)*. Association for Computing Machinery, New York, NY, USA, 3–14. https://doi.org/10.1145/3401025.3401731

[24] Ashraf Mahgoub, Karthick Shankar, Subrata Mitra, Ana Klimovic, Somali Chaterji, and Saurabh Bagchi. 2021. SONIC: Application-aware Data Passing for Chained Serverless Applications. In *2021 USENIX Annual Technical Conference* (Virtual Event, USA) *(USENIX ATC '21)*. USENIX Association, USA, 285–301.

[25] Modal. 2024. Modal. https://modal.com

[26] Ingo Müller, Renato Marroquín, and Gustavo Alonso. 2020. Lambada: Interactive Data Analytics on Cold Data Using Serverless Cloud Infrastructure. In *Proceedings of the 2020 ACM SIGMOD International Conference on Management of Data* (Portland, OR, USA) *(SIGMOD '20)*. Association for Computing Machinery, New York, NY, USA, 115–130. https://doi.org/10.1145/3318464.3389758

[27] Andrew Palmer, Prasad Phapale, Ilya Chernyavsky, Regis Lavigne, Dominik Fay, Artem Tarasov, Vitaly Kovalev, Jens Fuchser, Sergey Nikolenko, Charles Pineau, et al. 2017. FDR-controlled metabolite annotation for high-resolution imaging mass spectrometry. *Nature methods* 14, 1 (2017), 57–60.

[28] Qifan Pu, Shivaram Venkataraman, and Ion Stoica. 2019. Shuffling, fast and slow: scalable analytics on serverless infrastructure. In *Proceedings of the 16th USENIX Conference on Networked Systems Design and Implementation* (Boston, MA, USA) *(NSDI'19)*. USENIX Association, USA, 193–206.

[29] Shixiong Qi, Leslie Monis, Ziteng Zeng, Ian-chin Wang, and K. K. Ramakrishnan. 2022. SPRIGHT: extracting the server from serverless computing! high-performance eBPF-based event-driven, shared-memory processing. In *Proceedings of the ACM SIGCOMM 2022 Conference* (Amsterdam, Netherlands) *(SIGCOMM '22)*. Association for Computing Machinery, New York, NY, USA, 780–794. https://doi.org/10.1145/3544216.3544259

[30] Rredis. 2024. Redis. https://redis.io

[31] Josep Sampé, Gil Vernik, Marc Sánchez-Artigas, and Pedro García-López. 2018. Serverless Data Analytics in the IBM Cloud. In *Proceedings of the 19th International Middleware Conference Industry* (Rennes, France) *(Middleware '18)*. Association for Computing Machinery, New York, NY, USA, 1–8. https://doi.org/10.1145/3284028.3284029

[32] Marc Sánchez-Artigas and Germán T. Eizaguirre. 2022. A seer knows best: optimized object storage shuffling for serverless analytics. In *Proceedings of the 23rd ACM/IFIP International Middleware Conference* (Quebec, QC, Canada) *(Middleware '22)*. Association for Computing Machinery, New York, NY, USA, 148–160. https://doi.org/10.1145/3528535.3565241

[33] Marc Sánchez-Artigas, Germán T. Eizaguirre, Gil Vernik, Lachlan Stuart, and Pedro García-López. 2020. Primula: a Practical Shuffle/Sort Operator for Serverless Computing. In *Proceedings of the 21st International Middleware Conference Industrial Track* (Delft, Netherlands) *(Middleware '20)*. Association for Computing Machinery, New York, NY, USA, 31–37. https://doi.org/10.1145/3429357.3430522

[34] Marc Sánchez-Artigas and Pablo Gimeno Sarroca. 2021. Experience Paper: Towards enhancing cost efficiency in serverless machine learning training. In *Proceedings of the 22nd International Middleware Conference (Middleware '21)*. Association for Computing Machinery, New York, NY, USA, 210–222. https://doi.org/10.1145/3464298.3494884

[35] Vikram Sreekanti, Chenggang Wu, Xiayue Charles Lin, Johann Schleier-Smith, Joseph E. Gonzalez, Joseph M. Hellerstein, and Alexey Tumanov. 2020. Cloudburst: stateful functions-as-a-service. *Proc. VLDB Endow.* 13, 12 (2020), 2438–2452. https://doi.org/10.14778/3407790.3407836

[36] Bishoy Wadie, Lachlan Stuart, Christopher M. Rath, Bernhard Drotleff, Sergii Mamedov, and Theodore Alexandrov. 2024. METASPACE-ML: Metabolite annotation for imaging mass spectrometry using machine learning. *bioRxiv* (2024). https://doi.org/10.1101/2023.05.29.542736