# Trade-Offs and Challenges of Serverless Data Analytics


**Pedro García-López, Marc Sánchez-Artigas, Simon Shillaker, Peter Pietzuch, David Breitgand, Gil Vernik, Pierre Sutra, Tristan Tarrant, Ana Juan-Ferrer, and Gerard París**

**Abstract** Serverless computing has become very popular today since it largely simplifies cloud programming. Developers do no longer need to worry about provisioning or operating servers, and they have to pay only for the compute resources used when their code is run. This new cloud paradigm suits well for many applications, and researchers have already begun investigating the feasibility of serverless computing for data analytics. Unfortunately, today's serverless computing presents important limitations that make it really difficult to support all sorts of analytics workloads. This chapter first starts by analyzing three fundamental trade-offs of today's serverless computing model and their relationship with data analytics. It studies how by relaxing disaggregation, isolation, and simple scheduling, it is possible to increase the overall computing performance, but at the expense of essential aspects of the model such as elasticity, security, or sub-second activations, respectively. The consequence of these trade-offs is that analytics

P. García-López (✉) · M. Sánchez-Artigas · G. París
Universitat Rovira i Virgili, Tarragona, Spain
e-mail: pedro.garcia@urv.cat; marc.sanchez@urv.cat; gerard.paris@urv.cat

S. Shillaker · P. Pietzuch
Large Scale Data and Systems Group, Imperial College London, London, England
e-mail: s.shillaker17@imperial.ac.uk; prp@imperial.ac.uk

D. Breitgand · G. Vernik
Cloud Platforms, IBM Research Haifa, Haifa, Israel
e-mail: davidbr@il.ibm.com; gilv@il.ibm.com

P. Sutra
CNRS, Université Paris Saclay, Évry, France
e-mail: pierre.sutra@telecom-sudparis.eu

T. Tarrant
Red Hat, Cork, Ireland
e-mail: ttarrant@redhat.com

A. Juan-Ferrer
ATOS, Barcelona, Spain
e-mail: ana.juanf@atos.net



E. Curry et al. (eds.), *Technologies and Applications for Big Data Value*,
https://doi.org/10.1007/978-3-030-78307-5_3

applications may well end up embracing hybrid systems composed of serverless and serverful components, which we call *ServerMix* in this chapter. We will review the existing related work to show that most applications can be actually categorized as *ServerMix*.

**Keywords** Serverless computing · Data analytics · Cloud computing

## 1 Introduction

The chapter relates to the technical priority *Data Processing Architectures* of the European Big Data Value Strategic Research & Innovation Agenda [36]. It addresses the horizontal concerns *Data Analytics* and *The Cloud and HPC* of the BDV Technical Reference Model. The chapter relates to the *Systems, Methodologies, Hardware and Tools* cross-sectorial technology enablers of the AI, Data and Robotics Strategic Research, Innovation & Deployment Agenda [37].

With the emergence of serverless computing, the cloud has found a new paradigm that removes much of the complexity of its usage by abstracting away the provisioning of compute resources. This fairly new model was culminated in 2015 by Amazon in its Lambda service. This service offered cloud functions, marketed as FaaS (Function as a Service), and rapidly became the core of serverless computing. We say "core," because cloud platforms usually provide specialized serverless services to meet specific application requirements, packaged as BaaS (Backend as a Service). However, the focus of this chapter will be on the FaaS model, and very often, the words "serverless computing" and "FaaS" will be used interchangeably. The reason why FaaS drew widespread attention is because with FaaS platforms, a user-defined function and its dependencies are deployed to the cloud, where they are managed by the cloud provider and executed on demand. Simply put, users just write cloud functions in a high-level language and the serverless systems manage everything else: instance selection, auto-scaling, deployment, sub-second billing, fault tolerance, and so on. The programming simplicity of functions paves the way to soften the transition to the cloud ecosystem for end users.

Current practice shows that the FaaS model is well suited for many types of applications, provided that they require a small amount of storage and memory (see, for instance, AWS Lambda operational limits [3]). Indeed, this model was originally designed to execute event-driven, stateless functions in response to user actions or changes in the storage tier (e.g., uploading a photo to Amazon S3), which encompasses many common tasks in cloud applications. What was unclear is whether or not this new computing model could also be useful to execute data analytics applications. This question was answered partially in 2017 with the appearance of two relevant research articles: ExCamera [10] and "Occupy the Cloud" [19]. We say "partially," because the workloads that both works handled mostly consisted of "map"-only jobs, just exploiting embarrassingly massive parallelism. In particular, ExCamera proved to be 60% faster and 6x cheaper than using VM instances when encoding videos on the fly over thousands of Lambda functions.

The "Occupy the Cloud" paper showcased simple MapReduce jobs executed over Lambda Functions in their PyWren prototype. In this case, PyWren was 17% slower than PySpark running on `r3.xlarge` VM instances. The authors claimed that the simplicity of configuration and inherent elasticity of Lambda functions outbalanced the performance penalty. They, however, did not compare the costs between their Lambda experiments against an equivalent execution with virtual machines (VMs).

While both research works showed the enormous potential of serverless data analytics, today's serverless computing offerings importantly restrict the ability to work efficiently with data. In simpler terms, serverless data analytics are way more expensive and less performant than cluster computing systems or even VMs running analytics engines such as Spark. Two recent articles [17, 20] have outlined the major limitations of the serverless model in general. Remarkably, [20] reviews the performance and cost of several data analytics applications and shows that: a MapReduce-like sort of 100 TB was 1% faster than using VMs, but costing 15% higher; linear algebra computations [33] were 3x slower than an MPI implementation in a dedicated cluster, but only valid for large problem sizes; and machine learning (ML) pipelines were 3–5x faster than VM instances, but up to 7x higher total cost.

Furthermore, existing approaches must rely on auxiliary serverful services to circumvent the limitations of the stateless serverless model. For instance, PyWren [19] uses Amazon S3 for storage, coordination, and as indirect communication channel. Locus [28] uses Redis through the ElastiCache service, while ExCamera [10] relies on an external VM-based rendezvous and communication service. Also, Cirrus [7] relies on disaggregated in-memory servers.

The rest of the chapter is structured as follows. Section 1.1 presents the *ServerMix* model. Trade-offs of serverless architectures are analyzed in Sect. 2, while related work is revisited in Sect. 3. The challenges and advances in CloudButton project are presented in Sect. 4. Finally, Sect. 5 concludes the chapter.

## 1.1 On the Path to Serverless Data Analytics: The *ServerMix* Model

In the absence of a fully fledged serverless model in today's cloud platforms (e.g., there is no effective solution to the question of serverless storage in the market), current incarnations of serverless data analytics systems are hybrid applications combining serverless and serverful services. In this chapter, we identify them as "*ServerMix*." Actually, we will show how most related work can be classified under the umbrella term of *ServerMix*. We will first describe the existing design trade-offs involved in creating *ServerMix* data analytics systems. We will then show that it is possible to relax core principles such as disaggregation, isolation, and simple scheduling to increase performance, but also how this relaxation of the model may

compromise the auto-scaling ability, security, and even the pricing model and fast startup time of serverless functions. For example:

- **Relaxation of disaggregation:** Industry trends show a paradigm shift to disaggregated datacenters [12]. By physically decoupling resources and services, datacenter operators can easily customize their infrastructure to maximize the performance-per-dollar ratio. One such example of this trend is serverless computing. That is, FaaS offerings are of little value by themselves and need a vast ecosystem of disaggregated services to build applications. In the case of Amazon, this includes S3 (large object storage), DynamoDB (key-value storage), SQS (queuing services), SNS (notification services), etc. Consequently, departing from a serverless data-shipping model built around these services to a hybrid model where computations can be delegated to the stateful storage tier can easily achieve performance improvements [30]. However, disaggregation is the fundamental pillar of improved performance and elasticity in the cloud.
- **Relaxation of isolation:** Serverless platforms leverage operating system containers such as Docker to deploy and execute cloud functions. In particular, each cloud function is hosted in a separate container. However, functions of the same application may not need such a strong isolation and be co-located in the same container, which improves the performance of the application [1]. Further, cloud functions are not directly network-addressable in any way. Thus, providing direct communication between functions would reduce unnecessary latencies when multiple functions interact with one another, such that one function's output is the input to another one. Leveraging lightweight containers [26], or even using language-level constructs, would also reduce cold starts and boost inter-function communication. However, strong isolation and sandboxing is the basis for multi-tenancy, fault isolation, and security.
- **Flexible QoS and scheduling:** Current FaaS platforms only allow users to provision some amount of RAM and a time slice of CPU resources. In the case of Amazon Lambda, the first determines the other. Actually, there is no way to access specialized hardware or other resources such as the number of CPUs, GPUs, etc. To ensure service level objectives (SLOs), users should be able to specify resource requirements. But, this would lead to implement complex scheduling algorithms that were able to reserve such resources and even execute cloud functions in specialized hardware such as GPUs with different isolation levels. However, this would make it harder for cloud providers to achieve high resource utilization, as more constraints are put on function scheduling. Simple user-agnostic scheduling is the basis for short start-up times and high resource utilization.

It is clear that these approaches would obtain significant performance improvements. But, depending on the changes, such systems would be much closer to a serverful model based on VMs and dedicated resources than to the essence of serverless computing. In fact, we claim in this chapter that the so-called limitations of the serverless model are indeed its defining traits. When applications should require less disaggregation (computation close to the data), relaxation of

isolation (co-location, direct communication), or tunable scheduling (predictable performance, hardware acceleration), a suitable solution is to build a *ServerMix* solution. At least for serverless data analytics, we project that in the near future the dependency on serverful computing will increasingly "vanish," for instance, by the appearance of high-throughput, low-latency BaaS storage services, so that many *ServerMix* systems will eventually become 100% serverless. Beyond some technical challenges, we do not see any fundamental reason why pure serverless data analytics would not flourish in the coming years.

In the meantime, we will scrutinize the *ServerMix* model to provide a simplified programming environment, *as much closer as possible to serverless*, for data analytics. To this aim, under the context of the H2020 CloudButton project, we will work on the following three points: (i) Smart scheduling as a mechanism for providing transparent provisioning to applications while optimizing the cost-performance tuple in the cloud; (ii) fine-grained mutable state disaggregation built upon consistent state services; and (iii) lightweight and polyglot serverful isolation-novel lightweight serverful FaaS runtimes based on WebAssembly [15] as universal multi-language substrate.

## 2 Fundamental Trade-Offs of Serverless Architectures

In this section, we will discuss three fundamental trade-offs underpinning cloud functions architectures—packaged as FaaS offerings. Understand these trade-offs are important, not just for serverless data analytics but to open the minds of designers to a broader range of serverless applications. While prior works such as [17, 20] have already hinted these trade-offs, the contribution of this section is to explain in more detail that the incorrect navigation of these trade-offs can compromise essential aspects of the FaaS model.

The first question to ask is which are the essential aspects of the serverless model. For this endeavor, we will borrow the Amazon's definition of serverless computing, which is an unequivocal reference definition of this new technology. According to Amazon, the four characteristic features of a serverless system are:

- *No server management:* implies that users do not need to provision or maintain any servers
- *Flexible scaling:* entails that the application can be scaled automatically via units of consumption (throughput, memory) rather than units of individual servers
- *Pay for value:* is to pay for the use of consumption units rather than server units
- *Automated high availability:* ensures that the system must provide built-in availability and fault tolerance.

As we argue in this section, these four defining properties can be put in jeopardy but relaxing the tensions among three important architectural aspects that support them. These implementation aspects, which are *disaggregation*, *isolation*, and simple *scheduling*, and their associated trade-offs, have major implications on the

success of the FaaS model. In this sense, while a designer can decide to alter one or more of these trade-offs, for example, to improve performance, an oversimplifying or no comprehension of them can lead to hurt the four defining properties of the serverless model. Let us see how the trade-offs affect them.

## *2.1 Disaggregation*

Disaggregation is the idea of decoupling resources over high bandwidth networks, giving us independent resource pools. Disaggregation has many benefits, but importantly, it allows each component to (auto-)scale in an independent manner. In serverless platforms, disaggregation is the standard rather than an option, where applications are run using stateless functions that share state through disaggregated storage (e.g., such Amazon S3) [17, 20, 33]. This concept is backed up by the fact that modern high-speed networks allow for sub-millisecond latencies between the compute and storage layers—even allowing memory disaggregation like in InfiniSwap [14].

Despite the apparent small latencies, several works propose to relax disaggregation to favor performance. The reason is that storage hierarchies, across various storage layers and network delays, make disaggregation a bad design choice for many latency and bandwidth-sensitive applications such as machine learning [7]. Indeed, [17] considers that one of the limitations of serverless computing is its *data-shipping architecture*, where data and state are regularly shipped to functions. To overcome this limitation, the same paper proposes the so-called *fluid code and data placement* concept, where the infrastructure should be able to physically co-locate code and data. In a similar fashion, [2] proposes the notion of *fluid multi-resource disaggregation*, which consists of allowing movement (i.e., fluidity) between physical resources to enhance proximity, and thus performance. Another example of weakening disaggregation is [20]. In this paper, authors suggest to co-locate related functions in the same VM instances for fast data sharing.

Unfortunately, while data locality reduces data movements, it can hinder the elastic scale-out of compute and storage resources. In an effort to scale out wider and more elastically, processing mechanisms near the data (e.g., active storage [29]) have not been put at the forefront of cloud computing, though recently numerous proposals and solutions have emerged (see [18] for details). Further, recent works such as [30] show that active storage computations can introduce resource contention and interferences into the storage service. For example, computations from one user can harm the storage service to other users, thereby increasing the running cost of the application (pay for value). In any case, shipping code to data will interfere with the flexible scaling of serverless architectures due to the lack of fast and elastic datastore in the cloud [7].

Furthermore, ensuring locality for serverless functions would mean, for example, placing related functions in the same server or VM instance, while enabling fast shared memory between them. This would obviously improve performance in appli-

cations that require fast access to shared data such as machine learning and PRAM algorithms, OpenMP-like implementations of parallel algorithms, etc. However, as pinpointed in [20], besides going against the spirit of serverless computing, this approach would reduce the flexibility of cloud providers to place functions and consequently reduce the capacity to scale out while increasing the complexity of function scheduling. Importantly, this approach would force developers to think about low-level issues such as server management or whether function locality might lead suboptimal load balancing among server resources.

## *2.2 Isolation*

Isolation is another fundamental pillar of multi-tenant clouds services. Particularly, perfect isolation enables a cloud operator to run many functions (and applications) even on a single host, with low idle memory cost, and high resource efficiency. What cloud providers seek is to reduce the overhead of multi-tenant function isolation and provide high-performance (small startup times), for they leverage a wide variety of isolation technologies such as containers, unikernels, library OSes, or VMs. For instance, Amazon has recently released Firecracker microVMs for AWS Lambda, and Google has adopted gVisor. Other examples of isolation technologies for functions are CloudFlare Workers with WebAssembly or optimized containers such as SOCK [26]. These isolation techniques reduce startup times to the millisecond range, as compared to the second timescale of traditional VMs.

Beyond the list of sandboxing technologies for serverless computing, most of them battled-tested in the industry (e.g., Amazon Firecracker VMs), several research works have proposed to relax isolation in order to improve performance. For instance, [2] proposes the abstraction of a process in serverless computing, with the property that each process can be run across multiple servers. As a consequence of this multi-server vision, the paper introduces a new form of isolation that ensures multi-tenant isolation across multiple servers (where the functions of the same tenant are run). This new concept of isolation is called *coordinated isolation* in the paper. Further, [17] proposes two ways of relaxing isolation. The first one is based on the *fluid code and data placement* approach, and the second way is by allowing direct communication and network addressing between functions. In particular, the paper claims that today's serverless model stymies distributed computing due to its lack of direct communication among functions and advocates for long-running, addressable virtual agents instead.

Another technique to increase performance is to relax isolation and co-locate functions in the same VMs or containers [1, 20]. Co-location may be achieved using language-level constructs that reuse containers when possible. This can make sense for functions belonging to the same tenant [1], since it would heavily reduce cold starts and execution time for function compositions (or workflows). Unfortunately, it is possible that independent sets of sandboxed functions compete for the same server resources and interfere with each other's performance. Or simply, that it becomes

impossible to find a single host that have the necessary resources for running a sandbox of multiple functions, affecting important defining properties of serverless computing such as flexible scaling, pay for value, and no server management, among others.

Experiencing similar issues as above, it could be also possible to enable direct communication between functions of the same tenant. In this case, direct communication would permit a variety of distributed communication models, allowing, for example, the construction of replicated shared memory between functions. To put it baldly, each of these forms of relaxing isolation might in the end increase the attack surface, for instance, by opening physical co-residency attacks and network attacks not just to single functions but a collection of them.

## *2.3 Simple Scheduling*

Simple scheduling is another essential pillar of serverless computing. Indeed, cloud providers can ensure Quality of Service (QoS) and Service Level Agreements (SLAs) to different tenants by appropriately scheduling the reserved resources and bill them correspondingly. The goal of cloud scheduling algorithms is to maximize the utilization of the cloud resources while matching the requirements of the different tenants.

In today's FaaS offerings, tenants only specify the cloud function's memory size, while the function execution time is severely limited—for instance, AWS limits the execution time of functions to 15 min. This single constraint simplifies the scheduling of cloud functions and makes it easy to achieve high resource utilization through statistical multiplexing. For many developers, this lack of control on specifying resources, such as the number of CPUs, GPUs, or other types of hardware accelerators, is seen as an obstacle. To overcome this limitation, a clear candidate would be to work on more sophisticated scheduling algorithms that support more constraints on functions scheduling, such as hardware accelerators, GPUs, or the data dependencies between the cloud functions, which can lead to suboptimal function placement. For instance, it is not hard to figure out that a suboptimal placement of functions can result in an excess of communication to exchange data (e.g., for broadcast, aggregation, and shuffle patterns [20]) or in suboptimal performance. Ideally, these constraints should be (semi-)automatically inferred by the platform itself, for instance, from static code analysis, profiling, etc., to not break the property of "no server management," that is, the core principle of serverless. But even in this case, more constraints on function scheduling would make it harder to guarantee flexible scaling.

The literature also proposes ideas to provide predictable performance in serverless environments. For instance, [2] proposes the concept of "fine-grained live orchestration," which involves complex schedulers to allocate resources to serverless processes that run across multiple servers in the datacenter. Hellerstein et al. [17] advocates for heterogeneous hardware support for functions where developers

could specify their requirements in DSLs and the cloud providers would then calculate the most cost-effective hardware to meet user SLOs. This would guarantee the use of specialized hardware for functions. In [20], it is supported the claim of harnessing hardware heterogeneity in serverless computing. In particular, it is proposed that serverless systems could embrace multiple instance types (with prices according to hardware specs) or that cloud providers may select the hardware automatically depending on the code (like GPU hardware for CUDA code and TPU hardware for TensorFlow code).

Overall, the general observation is that putting more constraints on function scheduling for performance reasons could be disadvantageous in terms of flexible scaling and elasticity and even hinder high resource utilization. Moreover, it would complicate the pay per use model, as it would make it difficult to pay for the use of consumption units, rather than server units, due to hardware heterogeneity.

## *2.4 Summary*

As a summary, we refer to Fig. 1 as a global view of the overall trade-offs. These trade-offs have serious implications on the serverless computing model and require careful examination. As we have already seen, disaggregation, isolation, and simplified scheduling are pivotal to ensure flexible scaling, multi-tenancy, and millisecond startup times, respectively.

Weakening disaggregation to exploit function and data locality can be useful to improve performance. However, it also means to decrease the scale-out capacity of cloud functions and complicate function scheduling in order to meet user SLOs. The more you move to the left, the closer you are to serverful computing or running VMs or clusters in the datacenter.

With isolation the effect is similar. Since isolation is the key to multi-tenancy, completely relaxing isolation leaves nothing but dedicated resources. In your dedicated VMs, containers, or clusters (serverful), you can run functions very fast

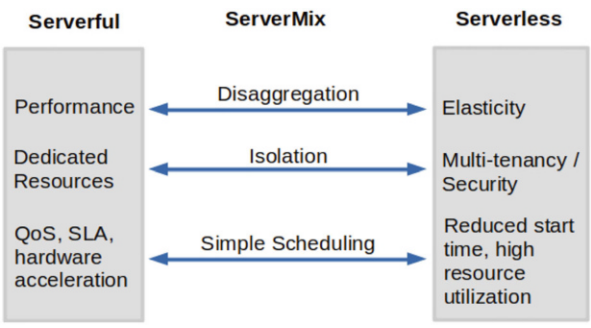


**Fig. 1** Trade-offs

without caring about sandboxing and security. But this also entails more complex scheduling and pricing models.

Finally, simple scheduling and agnostic function placement is also inherent to serverless computing. But if you require QoS, SLAs, or specialized hardware, the scheduling and resource allocation gets more complex. Again, moved to the extreme, you end up in serverful settings that already exist (dedicated resources, VMs, or clusters).

Perhaps, the most interesting conclusion of this figure is the region in the middle, which we call *ServerMix* computing. The zone in the middle involves applications that are built combining both serverless and serverful computing models. In fact, as we will review in the related work, many existing serverless applications may be considered *ServerMix* according to our definition.

## 3 Revisiting Related Work: The *ServerMix* Approach

### *3.1 Serverless Data Analytics*

Despite the stringent constraints of the FaaS model, a number of works have managed to show how this model can be exploited to process and transform large amounts of data [19, 21, 31], encode videos [10], and run large-scale linear algebra computations [33], among other applications. Surprisingly, and contrary to intuition, most of these serverless data analytics systems are indeed good *ServerMix* examples, as they combine both serverless and serverful components.

In general, most of these systems rely on an external, serverful provisioner component [10, 19, 21, 31, 33]. This component is in charge of calling and orchestrating serverless functions using the APIs of the chosen cloud provider. Sometimes the provisioner is called "coordinator" (e.g., as in ExCamera [10]) or "scheduler" (e.g., as in Flint [21]), but its role is the same: orchestrating functions and providing some degree of fault tolerance. But the story does not end here. Many of these systems require additional serverful components to overcome the limitations of the FaaS model. For example, recent works such as [28] use disaggregated in-memory systems such as ElastiCache Redis to overcome the throughput and speed bottlenecks of slow disk-based storage services such as S3. Or even external communication or coordination services to enable the communication among functions through a disaggregated intermediary (e.g., ExCamera [10]).

To fully understand the different variants of *ServerMix* for data analytics, we will review each of the systems one by one in what follows. Table 1 details which components are serverful and serverless for each system.

PyWren [19] is a proof of concept that MapReduce tasks can be run as serverless functions. More precisely, PyWren consists of a serverful function scheduler (i.e., a client Python application) that permits to execute "map" computations as AWS Lambda functions through a simple API. The "map" code to be run in parallel is

**Table 1** *ServerMix* applications

| | Components | |
|---|---|---|
| Systems | *Serverful* | *Serverless* |
| PyWren [19] | Scheduler | AWS Lambda, Amazon S3 |
| IBM PyWren [31] | Scheduler | IBM Cloud Functions, IBM COS, RabbitMQ |
| ExCamera [10] | Coordinator and rendezvous servers (Amazon EC2 VMs) | AWS Lambda, Amazon S3 |
| gg [11] | Coordinator | AW Lambda, Amazon S3, Redis |
| Flint [21] | Scheduler (Spark context on client machine) | AW Lambda, Amazon S3, Amazon SQS |
| Numpywren [33] | Provisioner, scheduler (client process) | AWS Lambda, Amazon S3, Amazon SQS |
| Cirrus [20] | Scheduler, parameter servers (large EC2 VM instances with GPUs) | AWS Lambda, Amazon S3 |
| Locus [28] | Scheduler, Redis service (AWS ElastiCache) | AWS Lambda, Amazon S3 |

first serialized and then stored in Amazon S3. Next, PyWren invokes a common Lambda function that deserializes the "map" code and executes it on the relevant datum, both extracted from S3. Finally, the results are placed back into S3. The scheduler actively polls S3 to detect that all partial results have been uploaded to S3 before signaling the completion of the job.

IBM-PyWren [31] is a PyWren derived project which adapts and extends PyWren for IBM Cloud services. It includes a number of new features, such as broader MapReduce support, automatic data discovery and partitioning, integration with Jupiter notebooks, and simple function composition, among others. For function coordination, IBM-PyWren uses RabbitMQ to avoid the unnecessary polling to the object storage service (IBM COS), thereby improving job execution times compared with PyWren.

ExCamera [10] performs digital video encoding by leveraging the parallelism of thousands of Lambda functions. Again, ExCamera uses serverless components (AWS Lambda, Amazon S3) and serverful ones (coordinator and rendezvous servers). In this case, apart from a coordinator/scheduler component that starts and coordinates functions, ExCamera also needs a rendezvous service, placed in an EC2 VM instance, to communicate functions among each other.

Stanford's gg [11] is an orchestration framework for building and executing burst-parallel applications over Cloud Functions. gg presents an intermediate representation that abstracts the compute and storage platform, and it provides dependency management and straggler mitigation. Again, gg relies on an external coordinator component, and an external Queue for submitting jobs (gg's thunks) to the execution engine (functions, containers).

Flint [21] implements a serverless version of the PySpark MapReduce framework. In particular, Flint replaces Spark executors by Lambda functions. It is similar

to PyWren in two main aspects. On the one hand, it uses an external serverful scheduler for function orchestration. On the other hand, it leverages S3 for input and output data storage. In addition, Flint uses Amazon's SQS service to store intermediate data and perform the necessary data shuffling to implement many of the PySpark's transformations.

Numpywren [33] is a serverless system for executing large-scale dense linear algebra programs. Once again, we observe the *ServerMix* pattern in numpywren. As it is based on PyWren, it relies on an external scheduler and Amazon S3 for input and output data storage. However, it adds an extra serverful component in the system called provisioner. The role of the provisioner is to monitor the length of the task queue and increase the number of Lambda functions (executors) to match the dynamic parallelism during a job execution. The task queue is implemented using Amazon SQS.

Cirrus machine learning (ML) project [20] is another example of a hybrid system that combines serverful components (parameter servers, scheduler) with serverless ones (AWS Lambda, Amazon S3). As with linear algebra algorithms, a fixed cluster size can either lead to severe underutilization or slowdown, since each stage of a workflow can demand significantly different amounts of resources. Cirrus addresses this challenge by enabling every stage to scale to meet its resource demands by using Lambda functions. The main problem with Cirrus is that many ML algorithms require state to be shared between cloud functions, for it uses VM instances to share and store intermediate state. This necessarily converts Cirrus into another example of a *ServerMix* system.

Finally, the most recent example of *ServerMix* is Locus [28]. Locus targets one of the main limitations of the serverless stateless model: data shuffling and sorting. Due to the impossibility of function-to-function communication, shuffling is ill-suited for serverless computing, leaving no other choice but to implement it through an intermediary cloud service, which could be cost-prohibitive to deliver good performance. Indeed, the first attempt to provide an efficient shuffling operation was realized in PyWren [19] using 30 Redis ElastiCache servers, which proved to be a very expensive solution. The major contribution of Locus was the development of a hybrid solution that considers both cost and performance. To achieve an optimal cost-performance trade-off, Locus combined a small number of expensive fast Redis instances with the much cheaper S3 service, achieving comparable performance to running Apache Spark on a provisioned cluster.

We did not include SAND [1] in the list of *ServerMix* systems. Rather, it proposes a new FaaS runtime. In the article, the authors of SAND present it as an alternative high-performance serverless platform. To deliver high performance, SAND introduces two relaxations in the standard serverless model: one in *disaggregation*, via a hierarchical message bus that enables function-to-function communication, and another in *isolation*, through application-level sandboxing that enables packing multiple application-related functions together into the same container. Although SAND was shown to deliver superior performance than Apache OpenWhisk, the paper failed to evaluate how these relaxations can affect the scalability, elasticity, and security of the standard FaaS model.

### *3.2 Serverless Container Services*

Hybrid cloud technologies are also accelerating the combination of serverless and serverful components. For instance, the recent deployment of Kubernetes (k8s) clusters in the big cloud vendors can help overcome the existing application portability issues in the cloud. There exists a plenty of hosted k8s services such as *Amazon Elastic Container Service* (EKS), *Google Kubernetes Engine* (GKE), and *Azure Kubernetes Service* (AKS), which confirm that this trend is gaining momentum. However, none of these services can be considered 100% "serverless." Rather, they should be viewed as a middle ground between cluster computing and serverless computing. That is, while these hosted services offload operational management of k8s, they still require custom configuration by developers. The major similarity to serverless computing is that k8s can provide short-lived computing environments like in the customary FaaS model.

But a very interesting recent trend is the emergence of the so-called serverless container services such as IBM Code Engine, AWS Fargate, Azure Container Instances (ACI), and Google Cloud Run (GCR). These services reduce the complexity of managing and deploying k8s clusters in the cloud. While they offer serverless features such as flexible automated scaling and pay-per-use billing model, these services still require some manual configuration of the right parameters for the containers (e.g., compute, storage, and networking) as well as the scaling limits for a successful deployment.

These alternatives are interesting for long-running jobs such as batch data analytics, while they offer more control over the applications thanks to the use of containers instead of functions. In any case, they can be very suitable for stateless, scalable applications, where the services can scale out by easily adding or removing container instances. In this case, the user establishes a simple CPU or memory threshold and the service is responsible for monitoring, load balancing, and instance creation and removal. It must be noted that if the service or application is more complex (e.g., a stateful storage component), the utility of these approaches is rather small, or they require important user intervention.

An important open source project related to serverless containers is CNCF's KNative. In short, KNative is backed by big vendors such as Google, IBM, and RedHat, among others, and it simplifies the creation of serverless containers over k8s clusters. Knative simplifies the complexity of k8s and Istio service mesh components, and it creates a promising substrate for both PaaS and FaaS applications.

**As a final conclusion**, we foresee that the simplicity of the serverless model will gain traction among users, so many new offerings may emerge in the next few years, thereby blurring the borders between both serverless and serverful models. Further, container services may become an interesting architecture for *ServerMix* deployments.

## 4 CloudButton: Towards Serverless Data Analytics

Serverless technologies will become a key enabler for radically simpler, user-friendly data analytics systems in the coming years. However, achieving this goal requires a programmable framework that goes beyond the current FaaS model and has user-adjustable settings to alter the IDS (Isolation-Disaggregation-Scheduling) trade-off (see Sect. 2 for more details)—for example, by weakening function isolation for better performance.

The EU CloudButton project [8] was born out of this need. It has been heavily inspired by "Occupy the Cloud" paper [19] and the statement made by a professor of computer graphics at UC Berkeley quoted in that paper:

> *"Why is there no cloud button?" He outlined how his students simply wish they could easily "push a button" and have their code—existing, optimized, single-machine code—running on the cloud."*

Consequently, our primary goal is *to create a serverless data analytics platform, which "democratizes Big Data" by overly simplifying the overall life cycle and cloud programming models* of data analytics. To this end, the 3-year CloudButton research project (2019–2021) will be undertaken as a collaboration between key industrial partners such as IBM, RedHat, and Atos, and academic partners such as Imperial College London, Institut Mines Télécom/Télécom SudParis, and Universitat Rovira i Virgili. To demonstrate the impact of the project, we target two settings with large data volumes: *bioinformatics* (genomics, metabolomics) and *geospatial data* (LiDAR, satellital), through institutions and companies such as EMBL, Pirbright Institute, Answare, and Fundación Matrix.

The project aims to provide full transparency [13] for applications which implies that we will be able to run unmodified single-machine code over effectively unlimited compute, storage, and memory resources thanks to serverless disaggregation and auto-scaling.

As we can see in Fig. 2, the CloudButton′s Lithops toolkit [32] will realize this vision of transparency relying on the next building blocks:

- **A high-performance serverless compute engine for Big Data:** The main goal is to support stateful and highly performant execution of serverless tasks. It will also provide efficient QoS management of containers that host serverless functions and a serverless execution framework to support typical dataflow models. As we can see in the Fig. 2, our design includes an extensible backend architecture for compute and storage that covers the major Cloud providers and Kubernetes cluster technologies.
- **Mutable, shared data support in serverless computing:** To simplify the transitioning from sequential to (massively-)parallel code, CloudButton has designed the Crucial [5] middleware on top of RedHat Infinispan that allows

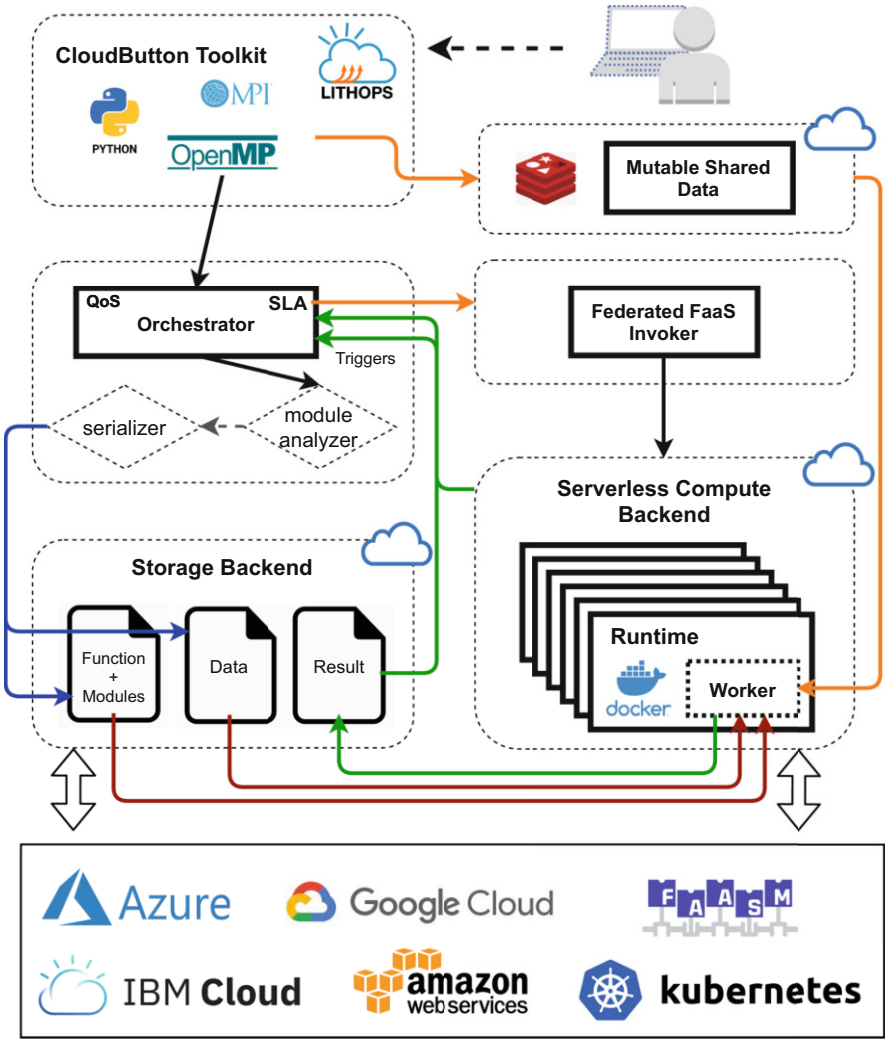


**Fig. 2** CloudButton architecture

the quickly spawning and easy sharing of mutable data structures in serverless platforms. This goal will explore the disaggregation area of the IDS trade-off.

- **Novel serverless cloud programming abstractions:** The goal is to express a wide range of existing data-intensive applications with minimal changes. The programming model should at the same time preserve the benefits of a serverless execution model and add explicit support for stateful functions in applications. Thanks to Lithops [32] and FaaSM [34] the toolkit will support almost unmodified data analytics applications in Python [32] but also C/C++ applications [34] using MPI and OpenMP programming models.

In what follows, we will delve deeper into each of these goals, highlighting in more detail the advance of each one with respect to the state of the art.

### *4.1 High-performance Serverless Runtime*

In many real-life cloud scenarios, enterprise workloads cannot be straightforwardly moved to a centralized public cloud due to the cost, regulation, latency and bandwidth, or a combination of these factors. This forces enterprises to adopt a hybrid cloud solution. However, current serverless frameworks are centralized and, out-of-the-box, they are unable to leverage computational capacity available in multiple locations.

Big Data analytics pipelines (a.k.a. analytics workflows) need to be efficiently orchestrated. There exists many serverless workflows orchestration tools (Fission flows, Argo, Apache Airflow), ephemeral serverless composition frameworks (IBM Composer), and stateful composition engines (Amazon Step Functions, Azure Durable Functions). To the best of our knowledge, workflow orchestration tools treat FaaS runtimes as black boxes that are oblivious to the workflow structure. A major issue with FaaS, which is exacerbated in a multi-stage workflow, is its data shipment architecture. Usually, the data is located in a separate storage service, such as Amazon S3 or IBM COS, and shipped for computation to the FaaS cluster. In general, FaaS functions are not scheduled with data locality in mind, even though data locality can be inferred from the workflow structure.

Further, and to the best of our knowledge, none of the existing workflow orchestration tools is serverless in itself. That is, the orchestrator is usually a stateful, always-on service. This is not necessarily the most cost-efficient approach for long running big data analytics pipelines, which might have periods of very high peakedness requiring massive parallelism interleaved with long periods of inactivity.

In CloudButton, we address the above challenges as follows:

- **Federated FaaS Invoker**: CloudButton exploits k8s federation architecture to provide a structured multi-clustered FaaS run time to facilitate analytics pipelines spanning multiple k8s clusters and Cloud Backends.
- **SLA, QoS, and scheduling**: programmers will be enabled to specify desired QoS levels for their functions. These QoS constraints will be enforced by a specialized scheduler (implemented via the k8s custom scheduler framework).
- **Serverless workflow orchestration**: we have constructed a Trigger-based orchestration framework [24] for *ServerMix* analytics pipelines. Tasks in the supported workflows can include massively parallel serverless computations carried out in Lithops.
- **Operational efficiency**: an operations cost-efficiency advisor will track the time utilization of each *ServerMix* component and submit recommendations on its appropriate usage.

### 4.2 Mutable Shared Data for Serverless Computing

In serverless Big Data applications, thousands of functions run in a short time. From a storage perspective, this requires the ability to scale abruptly the system to be on par with demand. To achieve this, it is necessary to decrease startup times (e.g., with unikernels [25]) and consider new directions for data distribution (e.g., Pocket [22]).

Current serverless computing platforms outsource state management to a dedicated storage tier (e.g., Amazon S3). This tier is agnostic of how data is mutated by functions, requiring serialization. This is cumbersome for complex data types, decreases code modularity and re-usability, and increases the cost of manipulating large objects. In contrast, we advocate that the storage tier should support in-place modifications. Additionally, storage requirements for serverless Big Data include:

- *Fast access (sub-millisecond) to ephemeral mutable data:* to support iterative and stateful computations (e.g., ML algorithms)
- *Fine-grained operations to coordinate concurrent function invocations*
- *Dependability:* to transparently support failures in both storage and compute tiers.

In CloudButton, we tackle these challenges by designing a novel storage layer for stateful serverless computation called Crucial [5]. Our goal is to simplify the transitioning from single-machine to massively parallel code. This requires new advances on data storage and distributed algorithms, such as:

- **Language support for mutable shared data.** The programmer can declare mutable shared data types in a piece of serverless code in a way transparently integrated to the programming language (e.g., with annotations). The storage tier knows the data types, allowing in-place mutations to in-memory shared data.
- **Data consistency.** Shared data objects are distributed and replicated across the storage layer, while maintaining strong consistency. To improve performance, developers can *degrade* data consistency [4, 35] on a per-object basis.
- **Just-right synchronization.** The implementation uses state machine replication atop a consensus layer [23, 27]. This layer self-adjusts to each shared data item, synchronizing replicas only when necessary, which improves performance.

### 4.3 Novel Serverless Cloud Programming Abstractions: The CloudButton Toolkit

Containers are the foundation of serverless runtimes, but the abstractions and isolation they offer can be restrictive for many applications. A hard barrier between the memory of co-located functions means all data sharing must be done via external storage, precluding data-intensive workloads and introducing an awkward programming model. Instantiating a completely isolated runtime environment for

each function is not only inefficient but at odds with how most language runtimes were designed.

This isolation boundary and runtime environment have motivated much prior work. A common theme is optimizing and modifying containers to better suit the task, exemplified by SOCK [26], which makes low-level changes to improve start-up times and efficiency. Others have partly sacrificed isolation to achieve better performance, for example, by co-locating a tenant's functions in the same container [1]. Also, a few frameworks for building serverless applications have emerged [10, 19, 21]. But these systems still require a lot of engineering effort to port existing applications.

Software fault isolation (SFI) has been proposed as an alternative isolation approach, offering memory-safety at low cost [6]. Introducing an intermediate representation (IR) to unify the spectrum of languages used in serverless has also been advocated [17]. WebAssembly is perfectly suited on both counts. It is an IR built on the principles of SFI, designed for executing multi-tenant code [16]. This is evidenced by its use in proprietary serverless technologies such as CloudFlare Workers and Fastly's Terrarium [9].

With the CloudButton toolkit, we build on these ideas and re-examine the serverless programming and execution environment. We have investigated new approaches to isolation and abstraction, focusing on the following areas:

- **Lightweight serverless isolation.** In the Faasm Backend [34], we combine SFI, WebAssembly, and existing OS tooling to build a new isolation mechanism, delivering strong security guarantees at a fraction of the cost of containers.
- **Efficient localized state.** This new isolation approach allows sharing regions of memory between co-located functions, enabling low-latency parallel processing and new opportunities for inter-function communication.
- **Stateful programming abstractions.** To make CloudButton programming seamless, we have created a new set of abstractions [5, 34], allowing users to combine stateful middleware with efficient localized state to easily build high-performance parallel applications.
- **Polyglot libraries and tooling.** By using a shared IR we can reuse abstractions across multiple languages. In this manner we will build a suite of generic tools to ease porting existing applications in multiple languages, including the CloudButton genomics and geospatial use-cases.

## 5 Conclusions and Future Directions

In this chapter, we have first analyzed three important architectural trade-offs of serverless computing: disaggregation, isolation, and simple scheduling. We have explained that by relaxing those trade-offs, it is possible to achieve higher performance, but also how that loosening can impoverish important serverless traits such as elasticity, multi-tenancy support, and high resource utilization. Moving the

trade-offs to the extremes, we have distinguished between serverful and serverless computing, and we have also introduced the new concept of *ServerMix* computing.

*ServerMix* systems combine serverless and serverful components to accomplish an analytics task. An ideal *ServerMix* system should keep resource provisioning transparent to the user and consider the cost-performance ratio as first citizen.

Finally, we have presented the CloudButton Serverless Data Analytics Platform and explained how it addresses the aforementioned trade-offs. CloudButton has demonstrated different levels of transparency for applications, enabling to run unmodified single-machine code over effectively unlimited compute, storage, and memory resources thanks to serverless disaggregation and auto-scaling. We predict that next-generation Cloud systems will offer a fully Serverless experience to users by combining both Serverless and Serverful infrastructures in a transparent way.

**Acknowledgments** This work has been partially supported by the EU project H2020 "CloudButton: Serverless Data Analytics Platform" (825184) and by the Spanish government (PID2019-106774RB-C22). Thanks also to the Serra Hunter programme from the Catalan government.

## References

1. Akkus, I. E., Chen, R., Rimac, I., Stein, M., Satzke, K., Beck, A., Aditya, P., & Hilt, V. (2018). SAND: Towards high-performance serverless computing. In 2018 USENIX annual technical conference (ATC'18), (pp. 923–935).
2. Al-Ali, Z., Goodarzy, S., Hunter, E., Ha, S., Han, R., Keller, E., & Rozner, E. (2018). Making serverless computing more serverless. In IEEE 11th international conference on cloud computing (CLOUD'18), (pp. 456–459).
3. Amazon: AWS lambda limits (2019). https://docs.aws.amazon.com/lambda/latest/dg/limits.html/
4. Attiya, H., & Welch, J. L. (1991). Sequential consistency versus linearizability (extended abstract). In Proceedings of the third annual ACM symposium on parallel algorithms and architectures (SPAA '91), (pp. 304–315).
5. Barcelona-Pons, D., Sánchez-Artigas, M., París, G., Sutra, P., & García-López, P. (2019). On the faas track: Building stateful distributed applications with serverless architectures. In Proceedings of the 20th international middleware conference (pp. 41–54).
6. Boucher, S., Kalia, A., Andersen, D. G., & Kaminsky, M. (2018). Putting the micro back in microservice. In 2018 USENIX annual technical conference (ATC '18) (pp. 645–650).
7. Carreira, J., Fonseca, P., Tumanov, A., Zhang, A. M., & Katz, R. (2018). A case for serverless machine learning. In: Workshop on systems for ML and open source software at NeurIPS.
8. H2020 CloudButton (2019) Serverless data analytics. http://cloudbutton.eu
9. Fastly: Fastly Labs—Terrarium (2019). https://www.fastlylabs.com/
10. Fouladi, S., Wahby, R. S., Shacklett, B., Balasubramaniam, K. V., Zeng, W., Bhalerao, R., Sivaraman, A., Porter, G., & Winstein, K. (2017). Encoding, fast and slow: Low-latency video processing using thousands of tiny threads. In Proceedings of the 14th USENIX symposium on networked systems design and implementation (NSDI'17) (pp. 363–376).
11. Fouladi, S., Romero, F., Iter, D., Li, Q., Chatterjee, S., Kozyrakis, C., Zaharia, M., & Winstein, K. (2019). From laptop to lambda: Outsourcing everyday jobs to thousands of transient functional containers. In 2019 USENIX Annual Technical Conference (ATC'19) (pp. 475–488).

12. Gao, P. X., Narayan, A., Karandikar, S., Carreira, J., Han, S., Agarwal, R., Ratnasamy, S., & Shenker, S. (2016). Network requirements for resource disaggregation. In Proceedings of the 12th USENIX conference on operating systems design and implementation (OSDI'16) (pp. 249–264).
13. García-López, P., Slominski, A., Shillaker, S., Behrendt, M., & Metzler, B. (2020). Serverless end game: Disaggregation enabling transparency. arXiv preprint arXiv:2006.01251.
14. Gu, J., Lee, Y., Zhang, Y., Chowdhury, M., & Shin, K.G. (2017). Efficient memory disaggregation with infiniswap. In 14th USENIX conference on networked systems design and implementation (NSDI'17) (pp. 649–667).
15. Haas, A., Rossberg, A., Schuff, D. L., Titzer, B. L., Holman, M., Gohman, D., Wagner, L., Zakai, A., & Bastien, J. (2017). Bringing the web up to speed with webassembly. In Proceedings of the 38th ACM SIGPLAN conference on programming language design and implementation (PLDI'17) (pp. 185–200).
16. Haas, A., Rossberg, A., Schuff, D. L., Titzer, B. L., Holman, M., Gohman, D., Wagner, L., Zakai, A., & Bastien, J. (2017). Bringing the web up to speed with WebAssembly. In Proceedings of the 38th ACM SIGPLAN conference on programming language design and implementation (PLDI'17) (pp. 185–200).
17. Hellerstein, J. M., Faleiro, J., Gonzalez, J. E., Schleier-Smith, J., Sreekanti, V., Tumanov, A., & Wu, C. (2019). Serverless computing: One step forward, two steps back. In Conference on innovative data systems research (CIDR'19).
18. Istvan, Z., Sidler, D., & Alonso, G. (2018). Active pages 20 years later: Active storage for the cloud. *IEEE Internet Computing, 22*(4), 6–14.
19. Jonas, E., Pu, Q., Venkataraman, S., Stoica, I., & Recht, B. (2017). Occupy the cloud: Distributed computing for the 99%. In Proceedings of the 2017 symposium on cloud computing (SoCC'17) (pp. 445–451).
20. Jonas, E., et al. (2019). *Cloud programming simplified: A Berkeley view on serverless computing*. https://arxiv.org/abs/1902.03383
21. Kim, Y., & Lin, J. (2018). Serverless data analytics with Flint. CoRR abs/1803.06354. http://arxiv.org/abs/1803.06354
22. Klimovic, A., Wang, Y., Stuedi, P., Trivedi, A., Pfefferle, J., & Kozyrakis, C. (2018). Pocket: Elastic ephemeral storage for serverless analytics. In Proceedings of the 13th USENIX symposium on operating systems design and implementation (OSDI'18) (pp. 427–444).
23. Lamport, L. (1998). The part-time parliament. *ACM Transactions on Computer Systems, 16*(2), 133–169. doi:http://doi.acm.org/10.1145/279227.279229
24. López, P. G., Arjona, A., Sampé, J., Slominski, A., Villard, L. (2020). Triggerflow: Trigger-based orchestration of serverless workflows. In: Proceedings of the 14th ACM international conference on distributed and event-based systems (pp. 3–14).
25. Manco, F., Lupu, C., Schmidt, F., Mendes, J., Kuenzer, S., Sati, S., Yasukata, K., Raiciu, C., & Huici, F. (2017). My VM is lighter (and safer) than your container. In Proceedings of the 26th symposium on operating systems principles (SOSP '17) (pp. 218–233).
26. Oakes, E., Yang, L., Zhou, D., Houck, K., Harter, T., Arpaci-Dusseau, A., & Arpaci-Dusseau, R. (2018). SOCK: Rapid task provisioning with serverless-optimized containers. In: 2018 USENIX annual technical conference (ATC'18) (pp. 57–70).
27. Ongaro, D., & Ousterhout, J.K. (2014). In search of an understandable consensus algorithm. In 2014 USENIX conference on USENIX annual technical conference (ATC'14) (pp. 305–319).
28. Pu, Q., Venkataraman, S., & Stoica, I. (2019). Shuffling, fast and slow: Scalable analytics on serverless infrastructure. In: Proceedings of the 16th USENIX symposium on networked systems design and implementation (NSDI'19) (pp. 193–206).
29. Riedel, E., Gibson, G. A., Faloutsos, C. (1998). Active storage for large-scale data mining and multimedia. In Proceedings of the 24rd international conference on very large data bases (VLDB'98) (pp. 62–73).
30. Sampé, J., Sánchez-Artigas, M., García-López, P., & París, G. (2017). Data-driven serverless functions for object storage. In Proceedings of the 18th ACM/IFIP/USENIX middleware conference (pp. 121–133). ACM, New York.

31. Sampé, J., Vernik, G., Sánchez-Artigas, M., & García-López, P. (2018). Serverless data analytics in the IBM Cloud. In Proceedings of the 19th international middleware conference industry, Middleware '18 (pp. 1–8). ACM, New York.
32. Sampé, J., García-López, P., Sánchez-Artigas, M., Vernik, G., Roca-Llaberia, P., & Arjona, A. (2021). Toward multicloud access transparency in serverless computing. *IEEE Software, 38*, 68–74.
33. Shankar, V., Krauth, K., Pu, Q., Jonas, E., Venkataraman, S., Stoica, I., Recht, B., & Ragan-Kelley, J. (2018). Numpywren: Serverless linear algebra. CoRR abs/1810.09679.
34. Shillaker, S., & Pietzuch, P. (2020). Faasm: Lightweight isolation for efficient stateful serverless computing. arXiv preprint arXiv:2002.09344.
35. Wada, H., Fekete, A., Zhao, L., Lee, K., & Liu, A. (2011). Data consistency properties and the trade-offs in commercial cloud storage: the consumers' perspective. In Fifth Biennial conference on innovative data systems research (CIDR'11) (pp. 134–143).
36. Zillner, S., Curry, E., Metzger, A., Auer, S., & Seidl, R. (Eds.) (2017). European big data value strategic research & innovation agenda. Big data value association
37. Zillner, S., et al. (Eds.) (2020). Strategic research, innovation and deployment Agenda—AI, data and robotics partnership. Third Release. BDVA, euRobotics, ELLIS, EurAI and CLAIRE.