Rethinking Serverless Computing: from the Programming Model to the Platform Design

Rethinking Serverless Computing: from the Programming Model to the Platform Design

Gustavo Alonso[1], Ana Klimovic[1], Tom Kuchler[1] and Michael Wawrzoniak[1]

[1]Systems Group, Computer Science Department, ETH Zürich, Switzerland

Abstract

Serverless computing offers a number of advantages over conventional, Virtual Machine (VM) based deployments on the cloud, e.g., greater elasticity, simplicity of use and management, finer granularity billing, and rapid deployment and start up times. Naturally, there is a growing interest in exploring how to run applications in this new environment and data analytics is not an exception. Unfortunately, current serverless platforms are limited along several dimensions, which makes things quite difficult from the perspective of data analytics. In this paper we explore what serverless has to offer today, what is missing, and what can be done to make serverless a better computing platform in general and for data analytics in particular.

Keywords

Cloud Computing, Serverless, Data Analytics, Functions as a Service

1. Introduction

Function as a Service (FaaS) or *serverless* represents an evolution of cloud computing services, where most of the complexities associated with deploying, starting, managing, maintaining, and retiring applications and their associated resources are hidden behind a much simpler interface. While the details of the commercial offerings differ somewhat [1, 2, 3] they all have several important aspects in common: they provide finer granularity billing than regular VMs, elasticity is managed automatically with support for launching a large number of functions concurrently, and the start-up time of the functions is significantly faster than that of VM-based applications.

In practice, however, serverless means different things depending on how one looks at it: as a user, as a cloud provider, and as a researcher. These perspectives are often not entirely aligned so we start by discussing each one of them. We then provide some background about data analytics on serverless before proposing an ambitious research agenda around serverless in general and data analytics on serverless in particular. This research agenda takes into account the way the cloud has been evolving and, importantly, the perspectives on the technology from both the user and the cloud provider, an essential aspect to make any proposal around serverless succeed. We propose a redesign of current serverless computing, from the programming model to the underlying system software stack, which can lead to significant performance and resource efficiency gains by freeing serverless from some of the legacy infrastructure it is built on today. With these ideas we put forward an alternative view on serverless that goes beyond data analytics but that can play a crucial role in making data analytics viable in the medium and long term on top of future serverless platforms. The view is based on our own research efforts in making serverless a viable and more efficient alternative to current cloud offerings and it encompasses all aspects of the stack from the serverless platform itself to the programming abstractions, interfaces, and support that are made available to applications.

Joint Workshops at 49th International Conference on Very Large Data Bases (VLDBW'23) — Workshop on Serverless Data Analytics (SDA'23), August 28 - September 1, 2023, Vancouver, Canada

alonso@inf.ethz.ch (G. Alonso); aklimovic@ethz.ch (A. Klimovic); tom.kuchler@inf.ethz.ch (T. Kuchler); michalw@inf.ethz.ch (M. Wawrzoniak)

0000-0002-4396-6695 (G. Alonso); 0000-0001-8559-0529 (A. Klimovic); 0009-0002-8091-0313 (T. Kuchler); 0000-0002-1304-8420 (M. Wawrzoniak)

2. Perspectives on Serverless

In this section we briefly discuss how users see serverless, what serverless represents for the provider, and different ways to tackle serverless from a research perspective. Here we focus on serverless platforms in general and discuss their current implementation on top of conventional cloud system software technology. In Section 3, we will revisit these concepts with data analytics in mind.

2.1. Serverless for the Users

From the perspective of the cloud user, for a narrow set of tasks, serverless is attractive as there is no need to deal with complex infrastructure or resource management. Functions run only as long as needed, thereby minimizing costs, and the automatic elasticity makes it easier to deal with a large number of parallel tasks, such as processing a collection of images or files in parallel. The event driven programming model behind serverless also makes it quite useful for doing things *on the side* of

larger deployments, providing a low cost and efficient way to extend existing functionality. A running application only needs to raise an event that triggers the creation of functions that perform a given task and then disappear when finished. Such a mechanism has been proposed as a way to, e.g., provide higher elasticity to data processing platforms [4] or to implement ephemeral caching services [5].

2.2. Serverless for the Provider

From the perspective of the cloud provider, serverless has other merits. On the one hand, it is an opportunity for the platform provider to optimize the infrastructure under the hood for better resource efficiency. As it is offered today, serverless is optimized for short, ephemeral tasks. Managing workloads composed of short (currently on AWS Lambda up to 15 minutes), simpler tasks allows the provider to fit these tasks on resources that would otherwise be too small to run larger jobs. One way to look at serverless from the provider perspective is as a way to fill in the cracks left by larger cloud application deployments.

On the other hand, since serverless exposes a higher level of abstraction of the cloud to users, this gives providers the opportunity to better manage the infrastructure by limiting the user's control over many of the aspects of their application's deployment. It is important to keep in mind that many applications and the way they run today in the cloud is a legacy of earlier years when users ran applications on their own servers rather than on cloud services. The option of renting VMs (Infrastructure as a Service) and many other existing services in the cloud are there solely so that users can port their application systems to the cloud. They are, by far, not the most efficient way to use the cloud. Serverless is a step towards removing part of that legacy and detaching the application from the underlying infrastructure (hence the name) so that the infrastructure can be better shared among multiple users, with better resource allocation, more control, tighter security, and less complexity to both the user and the provider. It is in this sense that serverless has been regarded as the next evolutionary step in cloud computing [6, 7].

2.3. Serverless for Researchers

From a research perspective, serverless has attracted attention from different areas, from systems to hardware to data management. Researchers in academia and industry have been improving the start up times of functions [8, 9, 10, 11], facilitating data sharing [12, 13, 14], building workflows of functions to implement more complex functionality [15, 16, 17], or enabling better hardware support for serverless [18]. These are all valuable ideas that will help to improve serverless as it evolves and also expand the use cases where serverless makes sense. However, serverless does not offer a seamless transition for applications such as data analytics.

As indicated above, current serverless offerings reflect many of the needs of cloud providers while still giving users an attractive new cloud service for a narrow set of use cases. As a result, researchers have pointed out severe limitations in current offerings that make running data analytics on serverless a major challenge [19]: lack of direct communication between functions, stateless functions, restricted life time, limited configuration choices between compute and memory, pricing model, etc. To deal with these limitations for serverless data analytics, two main approaches have emerged.

One approach tries to work around the current limitations. Examples of this line of work are Starling [20] and Lambada [21]. These two data processing systems demonstrate many of the complex techniques needed to make data analytics work on serverless: need to minimize data transfer through storage, speed up the launch of parallel functions at scale, overlap computation with I/O, etc. Yet, a message often overlooked from these efforts is that, from a cost point of view, serverless makes sense only at very low throughput (on the order of a few tens of queries per hour). For more intense workloads, it is cheaper to execute queries with standard data processing platforms on long-running VMs. The cost of serverless offerings is a crucial decision of the provider. We can only speculate why a unit of computation resources is significantly more expensive on serverless platforms than on conventional VMs. It could reflect the actual cost, in which case it does not look good for heavy data analytics, or it could be just a way to prevent cannibalizing the VM business line. Whatever it might be, this is an important factor to keep in mind at this stage before investing large efforts in implementing data analytic platforms on top of current serverless offerings.

Another approach tries to compensate for the limitations of serverless by building additional infrastructure that replaces missing functionality and provides the support needed to run data analytics as it is understood today. This includes creating caching layers to maintain state [22, 23], using proxies to facilitate better communication between functions [5], enabling limited forms of communication [24], improving scalability for concrete applications [25], etc. This work is often motivated by how serverless platforms are today and typically resorts to adding extra VM-based infrastructure for supporting functionality. To a certain extent, this goes against the goal of reducing infrastructure and facilitating management as those extra services are not typically managed by the serverless platform, but will have to be started, monitored, and turned off by the user. In an event-driven model, where the events triggering functions can appear

at any time, this burden may cancel out several of the potential advantages of serverless.

3. Background

Serverless is part of the cloud and, as such, it cannot be understood outside the context of cloud computing. Before presenting our view, we describe the aspects of cloud architecture relevant to the discussion here.

3.1. On the Cloud

Cloud computing has been evolving over the years towards higher levels of abstraction that hide more and more of the underlying computing infrastructure [26]. Hardware-as-a-Service, Platform-as-a-Service, Software-as-a-Service, Query-as-a-Service, and now Function-as-a-Service represent steps where what is being offered is an increasingly improved interface that frees the user from mundane administrative and management tasks: acquiring computers, maintaining and updating the operating system and supporting infrastructure like storage and file systems, deploying complex software and supporting its life cycle, etc.

This development is heavily influenced by the ability of customers to move their workloads to the cloud. Since applications evolve slower than hardware, it follows that the cloud cannot make radical changes in its architecture and the platforms it offers because otherwise users' applications would not run on the cloud or would have to be ported to the new environment made available by the provider. Such migration exercises are expensive and cumbersome so the cloud has always tried to provide a path to move applications with minimal changes. This is why VMs have played and still pay such a crucial role and why legacy systems like block storage or file systems are still provided since otherwise there are applications (notably database engines) that will simply not be able to operate in the cloud.

Yet, over the years, the cloud is evolving as the workloads are better understood, cloud native systems start to appear, and users get a better handle of running applications in the cloud. This evolution encompasses many aspects of the infrastructure: lightweight virtualization for more efficient bin-packing of securely-isolated tasks (e.g., Firecracker [8]), data representations suited to the disaggregated storage prevalent in the cloud (Parket, Arrow, etc.), supporting services native to the cloud (e.g., Key Value Stores as main memory caches), etc. The same can be said of the tools and systems the provider employs to manage and run the cloud as efficiently as possible.

3.2. On Serverless

Serverless is a next step in cloud offerings but, like all its predecessor services, it is necessarily bound by what is available on the provider side and the need to support an interface compatible with current applications. It provides a higher level of abstraction compared to renting some infrastructure (a machine, a VM, a piece of software) and it hides a great deal of the complexities of the cloud through automatic starts and shut-downs, automatic elasticity, finer granularity billing, etc. It also simplifies the choices available: instead of having to pick from an ever growing catalogue of machine configurations and sizes, current offerings limit the parameters available to the user (e.g., just memory size as in AWS Lambda). Finally, it also restricts the execution environment, allegedly to simplify the management of functions by the provider [7] by not allowing direct network communication among functions, enforcing data exchanges though storage services or queues, limiting the running time, not providing support for stateful services, etc.

Serverless functions today execute in MicroVMs (e.g., Firecracker [8]), which are lighter weight than regular VMs so that their start-up time is faster and they can be more densely bin-packed per machine. Functions are connected to regular cloud services in the form of storage, event management, or message queuing systems. However, these MicroVMs are still derived from their original VM counterparts and do not represent a significant departure from them in terms of functionality or interface. This is helpful for application compatibility but leaves significant opportunities for further performance and energy efficiency optimizations on the table. For example, MicroVMs still have significant startup times [27], context switching overheads [28], and memory duplication [29]. Given the significant differences in spirit and level of abstraction of serverless, it is worth revisitng whether the VM model is the correct one. As we will propose later, we think that it is not and a far more efficient approach is possible when the underlying support is built with a true serverless model in mind. This new approach might not be adequate for legacy applications but it will open up the way to implement *serverless native* applications.

3.3. On Cloud Data Processing

Relational database engines, one of the most successful forms of commercial software, provide the perfect example of the situation that the cloud creates. Conventional relational engines have an architecture designed decades ago and mostly focused on optimizing the data path from storage to the processor. Competitive commercial systems are built to run on the company's own servers and to take control of many aspects of the machine (mem-

ory management, data representation, I/O, scheduling, etc.) to the point that historically they have always been competing with the operating system for control of the machine. Making such engines run on the cloud so that users can port the stack running on top of the database has not been easy and has required to implement support for the legacy concepts that such engines need (e.g., block storage interfaces). As they are, database engines are almost the opposite of what the cloud is meant to offer: databases heavily depend on locality for performance, as the data anchors the engine to certain machines and data sources, the state that needs to be maintained is often huge making databases very cumbersome to migrate from one node to another and that same state makes databases very slow to start, the indirection layers of VMs and hypervisors often get in the way of database optimizations developed under the assumption there is nothing else running on the machine, etc. Not surprisingly, cloud providers often support custom systems to run legacy database engines (e.g., Amazon RDS).

As with many other services in the cloud, alternative designs have appeared that are better suited to the computing environment. Implicit in what we are discussing in the paper is the idea that data analytics refers to distributed query processing. In all fairness, that is not the only way to perform data analytics. There are the traditional database engines and data-warehouses, often monolithic engines with limited scalability, less elasticity, and not really suitable to the highly dynamic environment of serverless functions. As pointed out above, these engines already have issues when operating in the cloud and it is difficult to see how they could fit on top of serverless. Distributed query processing uses engines different from traditional ones (e.g., Snowflake [30]) or discards the notion of an engine entirely (e.g., Spark, Hadoop, or MapReduce can run queries at large scales but are not engines in the sense that they do not actually manage the data in the way a database engine does). The gap between legacy systems and what can be actually run on the cloud, together with the ever growing demand for data management support, has also led to a proliferation of cloud native data management and processing engines for a variety of applications and data types.

Serverless data analytics is at the same junction as database engines were with the cloud. Conventional data warehousing architectures are ill suited to the serverless model and, conversely, current serverless platforms are less than supportive of large scale data processing [19, 20, 21]. One can come up with many different ways to try to build analytics engines on serverless today but our own experience shows it is a never ending exercise with often only less than optimal options to choose from. If data analytics is ever going to run on serverless, a redesign from the ground up is necessary. In the next section we discuss how such a redesign could look like and its potential advantages over existing systems.

4. An Alternative View on Serverless

When considering the ideas above, it emerges that serverless, like the cloud in general, is likely to follow a path where it evolves towards something increasingly more optimized for the cloud but retains a certain degree of compatibility with existing systems. Thus, our alternative view on serverless aims at addressing two key research challenges that are particularly relevant for serverless analytics but also apply to serverless in general:

1. From a user perspective, how to leverage existing data analytics systems (e.g., Spark, Flink, Drill, Hadoop) that users are familiar with, and seamlessly run these engines with the main advantages of serverless: high elasticity, automated resource management, and fine-grain billing?
2. From a provider perspective, how to improve the performance and resource efficiency of the serverless infrastructure under the hood, which today is still rooted in a bloated system stack originally designed for long-running virtual machines?

These two questions are not orthogonal to each other. Addressing the first one by providing the means to run unmodified, existing distributed data processing systems on serverless would start to provide insights to inform future platform designs. It would also help identify which features of these systems collide with the serverless model. This information can then be used to change the way serverless is implemented and the interfaces the platform exposes to the application, so that the second challenge can be addressed while enabling a richer and more suitable interface to developers of serverless applications in general and serverless analytics in particular.

To address the first challenge, we propose to build an overlay system on top of the serverless platform that abstracts the FaaS infrastructure and provides the familiar POSIX-like environment of networked processes that existing distributed data analytics expect. This approach should enable running existing off-the-shelf data analytics engines on serverless platforms. The overlay system is responsible for filling the gap in functionality that the application expects (e.g., a distributed data engine like Spark expects to be able to exchange data directly over the network between workers) and what the underlying serverless platform supports (e.g., AWS Lambda and other commercial offerings today do not expose direct support for inter-function networking). The overlay will have to perform several tasks to turn current

serverless functions into something equivalent to regular VMs. Some of these tasks include: (1) intercepting system calls and redirecting them as needed so that application code that operates thinking it is running over an OS or a VM does not need to be modified; (2) establishing a group of communicating functions that will run the distributed application through services such a name and directory service, bootstrap processes, group coordination, etc.; and (3) provide basic fault tolerance on top of what the serverless platform provides to be able to react to issues caused by the platform itself, e.g., stragglers or functions failing. As we will discuss later, providing an initial version of such an interposition layer between the application and the serverless function already points out to interesting research directions such as the need to make I/O and communication more declarative so that the concrete implementation and runtime management can be done by the underlying serverless platform and not by the application as it is today. Section 5 explores these ideas and the prototype we are building in more detail.

Since the starting point is to better understand serverless analytics by enabling current systems to run on serverless platforms, we are addressing the first challenge by building a prototype of the overlay on top of a current commercial serverless offering (AWS Lambda). However, to address the second challenge (i.e., improve the performance and resource efficiency of serverless computing services), it is necessary to rethink the function execution model and system software stack of current serverless platforms. Hence, as a next step, we propose a new platform design. Importantly, by still leveraging the overlay on top of the platform, the changes we propose to the platform system software can be implemented transparently to user applications.

To address the second challenge, we propose to move away from AWS Lambda's approach of executing functions as MicroVMs. Instead, we advocate to treat serverless functions as true functions: bodies of code which take in a declared list of inputs and produce a list of outputs which can be fed to other functions or cloud data services. This enables a clear separation of computation (the function logic itself) and I/O (which can now be handled completely outside of the function by the platform, before and after the function's execution). Section 6 elaborates on the proposed platform design.

The separation of computation and I/O has several key advantages. First, compute tasks can be supported without the need for system calls, since the platform can prepare data for each function in a dedicated memory region before the function starts executing. Avoiding system calls removes a large attack surface of the untrusted user code and allows the platform to move away from the traditional VM-based isolation technologies, which introduce significant software bloat to provide secure isolation. Treating functions as true functions (which do not need to interact directly with the OS during their execution) enables adopting more lightweight sandbox designs, which can leverage emerging software and hardware isolation mechanisms, such as WebAssembly-based Software Fault Isolation [31, 32, 12], Memory Protection Keys (MPK) [33], and CHERI memory capabilities [34]. Second, separating compute and I/O directly exposes all dataflow to the platform, which enables optimizations like locality-aware function scheduling and overlapping data fetching and function execution. Finally, separating compute and I/O also makes functions more amenable to hardware acceleration, as pure compute tasks can more readily be accelerated on hardware platforms like GPUs and FPGAs. Meanwhile, the I/O tasks that the platform executes (i.e., to prepare data before function execution and to manage function outputs after execution) are good candidates to offload and accelerate on SmartNICs.

Note that this redesign of the serverless infrastructure is particularly interesting for data analytics. Queries can be easily represented as a dataflow (whether a tree or a DAG makes no difference) and the inputs and outputs of each stage are easily specified in a declarative manner. To a large extent, the serverless infrastructure can take advantage of the declarative nature of the I/O in functions to implement optimizations tasks that a traditional database engine performs: pre-fetching, caching, partial result reusing, view materialization, locality aware function scheduling, etc.; all tasks that are today missing from distributed data processing tools.

5. An Overlay on Serverless

The motivation to build an overlay on top of current serverless platforms arose from the experience accumulated trying to do query processing on serverless. While designing Lambada [21], looking at results from other groups [20], and examining attempts to run data heavy tasks on serverless [35, 36] we realized the big gap between what it is commercially available and what is needed to support data processing. Serverless works very well in narrow use cases with embarrassingly parallel tasks that can run independently of each other, have a short life time, do not exchange much data, and with a simple control flow between them (e.g., [37]). None of those are properties of analytic queries where operators create dependencies between the different stages of the computation, there is a well defined and potentially complex dataflow defining how and when to invoke the operators, and exchange potentially large amounts of intermediate results.

Focusing on the obvious limitation of today's offerings that forces data exchanges to occur through storage (S3 in AWS Lambda), we have developed an initial

prototype (Boxer 1.0) of the overlay that addressed basic communication between functions [38]. Using well known NAT-punching techniques, Boxer 1.0 was able to establish TCP-IP connections between functions and enabled us to implement distributed query processing as done in Lambada but without having to write and read to storage between stages of the query. This initial prototype showed the advantages of a more flexible serverless platforms and what could be gained by removing the underlying limitations rather than working around them.

Encouraged by that initial prototype, we have continued development towards Boxer 2.0, a more extensive implementation that incorporates not only the ability to communicate but also necessary services to, e.g., manage a distributed set of functions, ensure quick start up times of all the necessary functions through redundancy, better management of the function life cycle, etc. With the addition of an interposition layer that selectively intercepts C Library calls, the newer version of Boxer enables running off-the-shelf data processing engines like Apache Spark on top of serverless without any modifications.

We are in the process of porting a variety of data processing engines to Boxer 2.0. Doing so enables us to explore interesting ideas around serverless analytics like automatic instantiation of query engines on a per query basis [39]. But more importantly for the purposes here, Boxer 2.0 allows us to start running different query engines on serverless to identify how they need to be changed to take advantage of the unique features of serverless and also to study how the interface offered by serverless can be improved to better facilitate data analytic applications.

Although our first step is prototyping Boxer on top of existing commercial serverless platforms, such as AWS Lambda, we next plan to explore how to continue providing the familiar abstractions that traditional data analytics engines expect while evolving the underlying platform under the hood for greater performance and resource efficiency.

6. A New Serverless Platform

A serverless platform must meet several requirements from a provider perspective, requirements that also have an impact on how the platform is perceived by users:

Secure isolation: The platform must prevent untrusted user code from tampering with the infrastructure or accessing the data or code of other users.[1]

Low latency: A function should complete with low (ideally $\leq 10\%$) overhead compared to its execution on a dedicated, bare-metal server.

[1]We assume the typical cloud computing threat model [40], in which the provider does not trust users, users do not trust each other, but *users trust the cloud provider.*

High throughput per machine: The platform should serve a high rate of function invocations per physical machine (i.e., dense bin-packing) to maximize throughput at low cost.

The current platforms offered by cloud providers like AWS, Google, and Microsoft Azure try to satisfy these requirements based on lightweight VMs or secure containers [8, 41]. As described in Section 3.2, this approach leads to a bloated system software stack rooted in a more conventional cloud execution model based on long-running virtual machines and processes. Current platforms introduce high overhead when executing many fine-grain, short-lived tasks at high churn [12].

To optimize function execution latency and throughput while still satisfying the secure isolation requirement, we propose to rethink the serverless function execution model. Our key idea is to *treat functions as true functions*, which consist of a declarative list of inputs and outputs as well as the actual computation logic that operates on the inputs. This delcarative model enables a strict separation of computation and I/O. With this clear separation, functions now consist of pure computations and I/O is handled by the platform before/after function execution. The platform provides each function with a dedicated memory region prepared with the function's inputs and the function does not do any I/O (or in fact any system calls) during its execution. Avoiding system calls removes a large attack surface (VMs are used today as an isolation mechanism since the OS itself is considered too large of a TCB to securely isolate functions from untrusted users [8, 41]) and allows us to rethink function sandbox design. With our new execution model, we can leverage lighter weight isolation mechanisms optimized for performance (e.g., WASM [31], MPK [33], and CHERI memory capabilities [34]) rather than defaulting to the legacy approach of executing each function in a separate virtual machine, bundled with its own operating system.

We are currently building a prototype of a serverless platform, *Dandelion*, which adopts this declarative function execution model. We design the system software to efficiently schedule functions and execute them with support for a variety of hardware and software isolation mechanisms under the hood through a unified abstraction. In our initial prototype, we leverage CHERI memory capabilities [34] to isolate function memory regions while running the Dandelion system infrastructure and compute functions within a single virtual address space. We compare Dandelion's performance to a Firecracker system running functions in separate MicroVMs. We sweep function invocations for each system, with each function executing a matrix multiplication computation. Our initial results show that Dandelion achieves 3.4× higher peak throughput per machine and over 14× lower latency compared to Firecracker.

In addition to enabling lightweight function sandbox

technology for low latency and high-density bin-packing without compromising secure isolation, the declarative function execution model offers cloud providers additional opportunities to optimize the underlying infrastructure. A major benefit of declaring applications as compositions of compute functions (containing untrusted user code) and I/O tasks (handled by the platform) is that application dataflow becomes explicit and directly exposed to the platform. The cloud provider can leverage information about how data flows between functions and cloud data services to prefetch function inputs and avoid functions wasting CPU cycles while waiting for data by guaranteeing that a function only starts executing when its inputs are available [42]. The provider can also leverage dataflow information in applications for locality-aware scheduling, i.e., collocating functions that exchange data on the same nodes.

Finally, the declarative function execution model and strict separation of compute and I/O also lends itself well to hardware acceleration. User functions become easier to offload to hardware accelreators such as GPUs when they are pure functions that do not rely on close interaction with the host OS. We also plan to explore offloading the I/O tasks performed by the platform to SmartNICs. We believe this declarative execution model under the hood of a serverless platform is a promising way to leverage heterogeneous hardware, while the overlay presented in Section 5 can maintain a convenient abstraction for user applications.

7. Discussion

In this section we explore how the ideas just proposed could influence serverless data analytics. We also discuss the interplay between an overlay system like Boxer and our ideas around Dandelion about how to optimize the underlying serverless infrastructure.

7.1. Engines, Platforms, and Services

What we are proposing, even in the initial use case where we run existing, unmodified query platforms on top of serverless functions, brings query processing closer to the notion of query-as-a-service. As discussed elsewhere [39], the approach makes it possible to create a distributed query engine when a query arrives and dismantle it when it finishes. The resulting functionality is very similar to that of services such as Amazon Athena [43] or Google Big Query [44] except that the user can select the engine where the query runs.

When considering the modified serverless platform we have in mind, the opportunity arises to turn the serverless platform into an actual data processing engine providing much of the functionality missing on distributed query processing systems today. Current systems do not run on an actual engine in the sense that systems such as Spark do not maintain permanent data structures and functionality (indexes, lock tables, access statistics, schema, views, access controls, etc.) across executions. Thus, they do not perform common database optimizations.

In addition to running off-the-shelf engines on the overlay, an optimized platform like Dandelion enables building data processing engines directly on top of the declarative function execution model interface, as opposed to through the overlay. Through the declarative specification of I/O, function scheduling can be enhanced with features such as data pre-fetching from storage if that is where the data resides so when the function starts, the data has already been brought to it. This would significantly speed up running, e.g., a join of a base table with the intermediate results produced by another join of other tables. Similarly, it should be possible to implement different query execution models beyond the batch mode enforced by today's serverless platforms. Direct communications should enable vectorized and even volcano style execution models that are far more suitable to modern analytics and have much better performance. Providing such functionality would make serverless an excellent vehicle to implement a completely new generation of data analytic engines where the system functionality is directly embedded in the computational model. Such an architecture is likely to improve performance while allowing the provider to optimize internal bottlenecks in the cloud like the latency of accessing disaggregated storage or network congestion.

7.2. System Optimizations

Data analytics has many decades of experience on optimizing queries, predicting cardinalities, calculating costs of operations, etc. Many of these advantages are lost when operating on the distributed query processing settings found in the cloud today. This leads, in several subtle ways, to a number of inefficiencies that need to be addressed. Among them, the most relevant is overprovisioning: reserving a much larger set of resources than actually needed just in case, e.g., a load spike arrives. Overprovisioning affects all aspects of the system: memory, CPU capacity, number of machines, etc.

For instance, *stranded memory* [45] results from machines running out of virtual CPUs to allocate before they run out of memory. It is tempting to think that the current configuration of functions based solely on the memory size could be related to this provider-side problem. In the serverless data analytics view we propose, the problem could be addressed differently: apply traditional query optimization techniques and knowledge of the operators to estimate the amount of memory a function will need to execute. This would yield a far more accurate

allocation of resources on a per-query basis and, as a result, a lower budget for running queries than when the size of the functions is chosen on a worst case scenario basis.

Similarly, the notion of declarative interfaces at the function level can be used beyond I/O and be applied to networking and communication. Rather than using sockets as the current Boxer prototypes does, functions would specify a number of other functions and the required topology that the underlying system would automatically instantiate as needed. This opens up the opportunity for the provider to optimize the location of the functions, reuse them when possible, collocate them in the same machine if the opportunity arises, etc. It is not difficult to see how this would benefit parallel query execution and minimize data movement while still preserving the elasticity of serverless and hiding of the underlying infrastructure form the user.

Finally, the declarative nature of functions we propose would support the automatic creation and management of caching layers. The system can observe the declarations of function compositions and and decide which data can be moved to a faster storage layer (whether in memory, on a dedicated key value store, or even in accelerated storage) for faster access if used often enough.

7.3. Next Steps

The transition from the current Boxer overlay system to a radically different function execution system like Dandelion will involve exploring how to develop applications under the new interface. In our current prototype of Dandelion, we develop a domain specific language for developers to directly express their applications as compositions of pure compute and platform library I/O functions. However, in the future, we aim to explore how off-the-shelf applications can be automatically transpiled to compositions that strictly separate their compute and I/O tasks under the hood.

Running arbitrary legacy applications on Dandelion can be a challenge but less so for data processing and data analytics. Note that the declarative I/O model closely resembles the execution model of database engines with operators implementing the logic to be applied to supplied by a pre-defined execution model and interface (i.e., volcano, vectorized, or batched). To a first approximation, queries can be expressed as a DAG of functions that are connected through well defined I/O interfaces. An interesting research direction is to define such interfaces so that they fulfill the requirements of the serverless infrastructure while facilitating the execution of distributed query processing. Nevertheless, the ultimate goal of our vision for serverless is to have native engines that take full advantage of the new model rather than being anchored on architectures more suitable for conventional deployments on servers or VMs.

8. Conclusions

In this short paper we have put forward an alternative view on serverless that accommodates the perspectives of both the users and the providers. It defines a path of migration for existing distributed query processing systems to serverless and proposes a redesign of the interface offered to function code that truly hides the underlying execution platform. The idea is not exclusive to data analytics but we have show how data processing, even being an application ill suited for serverless, can greatly benefit from this alternative view.

The systems we are developing should help to inform how to better support data analytics on serverless by enabling running off-the-shelf systems on today's platforms. At the same time, the experience gathered from these initial experiments will help better define what the serverless infrastructure should become, to eliminate a lot of the legacy and bloat present in current systems, and provide an interface that unleashes all the potential of serverless as the next generation of cloud computing.

References

[1] Google Cloud Functions, 2023. URL: https://cloud.google.com/functions, visited 2023-8-9.

[2] AWS Lambda, 2023. URL: https://aws.amazon.com/lambda, visited 2023-08-9.

[3] Microsoft Azure Functions, 2023. URL: https://azure.microsoft.com/en-us/services/functions, visited 2023-08-9.

[4] H. Bian, T. Sha, A. Ailamaki, Using cloud functions as accelerator for elastic data analytics, Proc. ACM Manag. Data 1 (2023).

[5] A. Wang, J. Zhang, X. Ma, A. Anwar, L. Rupprecht, D. Skourtis, V. Tarasov, F. Yan, Y. Cheng, Infinicache: Exploiting ephemeral serverless functions to build a cost-effective memory cache, in: USENIX FAST, 2020.

[6] P. Castro, V. Ishakian, V. Muthusamy, A. Slominski, The rise of serverless computing, Commun. ACM 62 (2019) 44–54.

[7] J. Schleier-Smith, V. Sreekanti, A. Khandelwal, J. Carreira, N. J. Yadwadkar, R. A. Popa, J. E. Gonzalez, I. Stoica, D. A. Patterson, What serverless computing is and should become: The next phase of cloud computing, Commun. ACM 64 (2021) 76–84.

[8] A. Agache, M. Brooker, A. Iordache, A. Liguori, R. Neugebauer, P. Piwonka, D.-M. Popa, Firecracker: Lightweight virtualization for serverless applications, in: NSDI, 2020.

[9] E. Oakes, L. Yang, D. Zhou, K. Houck, T. Harter, A. Arpaci-Dusseau, R. Arpaci-Dusseau, SOCK: Rapid task provisioning with serverless-optimized containers, in: USENIX ATC, 2018.

[10] D. Du, T. Yu, Y. Xia, B. Zang, G. Yan, C. Qin, Q. Wu, H. Chen, Catalyzer: Sub-millisecond startup for serverless computing with initialization-less booting, in: ASPLOS, 2020.

[11] F. Manco, C. Lupu, F. Schmidt, J. Mendes, S. Kuenzer, S. Sati, K. Yasukata, C. Raiciu, F. Huici, My vm is lighter (and safer) than your container, in: SOSP, 2017.

[12] S. Shillaker, P. Pietzuch, Faasm: Lightweight isolation for efficient stateful serverless computing, in: 2020 USENIX Annual Technical Conference (USENIX ATC 20), 2020.

[13] A. Klimovic, Y. Wang, P. Stuedi, A. Trivedi, J. Pfefferle, C. Kozyrakis, Pocket: Elastic ephemeral storage for serverless analytics, in: OSDI, 2018, p. 427–444.

[14] M. Yu, T. Cao, W. Wang, R. Chen, Following the data, not the function: Rethinking function orchestration in serverless computing, in: 20th USENIX Symposium on Networked Systems Design and Implementation (NSDI 23), 2023.

[15] A. Mahgoub, E. B. Yi, K. Shankar, S. Elnikety, S. Chaterji, S. Bagchi, ORION and the three rights: Sizing, bundling, and prewarming for serverless DAGs, in: 16th USENIX Symposium on Operating Systems Design and Implementation (OSDI 22), 2022.

[16] A. Mahgoub, K. Shankar, S. Mitra, A. Klimovic, S. Chaterji, S. Bagchi, SONIC: Application-aware data passing for chained serverless applications, in: USENIX Annual Technical Conference (USENIX ATC 21), 2021.

[17] V. M. Bhasi, J. R. Gunasekaran, P. Thinakaran, C. S. Mishra, M. T. Kandemir, C. Das, Kraken: Adaptive container provisioning for deploying dynamic dags in serverless platforms, in: Proceedings of the ACM Symposium on Cloud Computing, SoCC '21, 2021, p. 153–167.

[18] D. Du, Q. Liu, X. Jiang, Y. Xia, B. Zang, H. Chen, Serverless computing on heterogeneous computers, in: Proceedings of the 27th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, ASPLOS '22, 2022.

[19] J. M. Hellerstein, J. M. Faleiro, J. Gonzalez, J. Schleier-Smith, V. Sreekanti, A. Tumanov, C. Wu, Serverless computing: One step forward, two steps back, in: CIDR, 2019.

[20] M. Perron, R. Castro Fernandez, D. DeWitt, S. Madden, Starling: A scalable query engine on cloud functions, in: SIGMOD, 2020.

[21] I. Müller, R. Marroquín, G. Alonso, Lambada: Interactive data analytics on cold data using serverless cloud infrastructure, in: SIGMOD, 2020.

[22] C. Wu, J. M. Faleiro, Y. Lin, J. M. Hellerstein, Anna: A KVS for Any Scale, in: ICDE, 2018.

[23] F. Romero, G. I. Chaudhry, I. n. Goiri, P. Gopa, P. Batum, N. J. Yadwadkar, R. Fonseca, C. Kozyrakis, R. Bianchini, Faa$t: A transparent auto-scaling cache for serverless applications, in: Proceedings of the ACM Symposium on Cloud Computing, SoCC '21, Association for Computing Machinery, New York, NY, USA, 2021, p. 122–137.

[24] S. Thomas, L. Ao, G. M. Voelker, G. Porter, Particle: Ephemeral endpoints for serverless networking, in: Proceedings of the 11th ACM Symposium on Cloud Computing, 2020, p. 16–29.

[25] Z. Jia, E. Witchel, Nightcore: Efficient and scalable serverless computing for latency-sensitive, interactive microservices, in: Proceedings of the 26th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, 2021, p. 152–166.

[26] M. Armbrust, A. Fox, R. Griffith, A. D. Joseph, R. Katz, A. Konwinski, G. Lee, D. Patterson, A. Rabkin, I. Stoica, M. Zaharia, A view of cloud computing, Commun. ACM 53 (2010) 50–58.

[27] D. Ustiugov, P. Petrov, M. Kogias, E. Bugnion, B. Grot, Benchmarking, analysis, and optimization of serverless function snapshots, in: Proceedings of the 26th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, ASPLOS '21, 2021, p. 559–572.

[28] N. C. Wanninger, J. J. Bowden, K. Shetty, A. Garg, K. C. Hale, Isolating functions at the hardware limit with virtines, in: Proceedings of the Seventeenth European Conference on Computer Systems, EuroSys '22, 2022.

[29] D. Saxena, T. Ji, A. Singhvi, J. Khalid, A. Akella, Memory deduplication for serverless computing with medes, Proceedings of the Seventeenth European Conference on Computer Systems (2022).

[30] M. Vuppalapati, J. Miron, R. Agarwal, D. Truong, A. Motivala, T. Cruanes, Building an elastic query engine on disaggregated storage, in: Proceedings of the 17th Usenix Conference on Networked Systems Design and Implementation, NSDI'20, USENIX Association, USA, 2020, p. 449–462.

[31] Webassembly, 2023. Available at https://webassembly.org, visited 2023-8-9.

[32] K. Varda, Webassembly on cloudflare workers, https://blog.cloudflare.com/webassembly-on-cloudflare-workers, 2018.

[33] S. Park, S. Lee, W. Xu, H. Moon, T. Kim, libmpk: Software abstraction for intel memory protection keys (intel MPK), in: 2019 USENIX Annual Technical Conference (USENIX ATC 19), 2019.

[34] R. N. Watson, J. Woodruff, P. G. Neumann, S. W. Moore, J. Anderson, D. Chisnall, N. Dave, B. Davis, K. Gudka, B. Laurie, S. J. Murdoch, R. Norton, M. Roe, S. Son, M. Vadera, Cheri: A hybrid capability-system architecture for scalable software compartmentalization, in: 2015 IEEE Symposium on Security and Privacy, 2015.

[35] J. Jiang, S. Gan, Y. Liu, F. Wang, G. Alonso, A. Klimovic, A. Singla, W. Wu, C. Zhang, Towards demystifying serverless machine learning training, in: Proceedings of the 2021 International Conference on Management of Data, 2021, p. 857–871.

[36] Y. Wu, T. T. A. Dinh, G. Hu, M. Zhang, Y. M. Chee, B. C. Ooi, Serverless data science - are we there yet? A case study of model serving, in: SIGMOD '22: International Conference on Management of Data, Philadelphia, PA, USA, June 12 - 17, 2022, 2022.

[37] S. Fouladi, F. Romero, D. Iter, Q. Li, S. Chatterjee, C. Kozyrakis, M. Zaharia, K. Winstein, From laptop to lambda: Outsourcing everyday jobs to thousands of transient functional containers, in: USENIX ATC, 2019.

[38] M. Wawrzoniak, I. Müller, R. Bruno, G. Alonso, Boxer: Data analytics on network-enabled serverless platforms, in: CIDR, 2021.

[39] M. Wawrzoniak, R. Bruno, A. Klimovic, G. Alonso, Ephemeral per-query engines for serverless analytics, in: 1st Workshop on Serverless Data Analytics (SDA'23) at VLDB 2023, 2023.

[40] Amazon Web Services, Security overview of AWS Lambda, https://docs.aws.amazon.com/whitepapers/latest/security-overview-aws-lambda/security-overview-aws-lambda.html, 2021.

[41] E. G. Young, P. Zhu, T. Caraza-Harter, A. C. Arpaci-Dusseau, R. H. Arpaci-Dusseau, The true cost of containing: A gVisor case study, in: 11th USENIX Workshop on Hot Topics in Cloud Computing (HotCloud 19), 2019.

[42] Y. Deng, A. Montemayor, A. Levy, K. Winstein, Computation-centric networking, in: Proceedings of the 21st ACM Workshop on Hot Topics in Networks, HotNets '22, 2022.

[43] Amazon Athena, 2023. URL: http://docs.aws.amazon.com/athena/, visited 2023-08-9.

[44] Google BigQuery, 2023. URL: https://cloud.google.com/bigquery/, visited 2023-08-9.

[45] H. Li, D. S. Berger, L. Hsu, D. Ernst, P. Zardoshti, S. Novakovic, M. Shah, S. Rajadnya, S. Lee, I. Agarwal, M. D. Hill, M. Fontoura, R. Bianchini, Pond: Cxl-based memory pooling systems for cloud platforms, in: Proceedings of the 28th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 2, ASPLOS 2023, 2023.