Le Taureau: Deconstructing the Serverless Landscape & A Look Forward

Anurag Khandelwal
Yale University

Arun Kejariwal
Facebook Inc.

Karthik Ramasamy
Splunk Inc.

ABSTRACT

Akin to the natural evolution of programming in assembly language to high-level languages, serverless computing represents the next frontier in the evolution of cloud computing: *bare metal → virtual machines → containers → serverless.* The genesis of serverless computing can be traced back to the fundamental need of enabling a programmer to singularly focus on writing application code in a high-level language and isolating all facets of system management (for example, but not limited to, instance selection, scaling, deployment, logging, monitoring, fault tolerance and so on). This is particularly critical in light of today's, increasingly tightening, time-to-market constraints. Currently, serverless computing is supported by leading public cloud vendors, such as AWS Lambda, Google Cloud Functions, Azure Cloud Functions and others. While this is an important step in the right direction, there are many challenges going forward. For instance, but not limited to, how to enable support for dynamic optimization, how to extend support for stateful computation, how to efficiently bin-pack applications, how to support hardware heterogeneity (this will be key especially in light of the emergence of hardware accelerators for deep learning workloads).

Inspired by Picasso's *Le Taureau*[1], in the tutorial proposed herein, we shall deconstruct evolution of serverless – the overarching intent being to facilitate better understanding of the serverless landscape. This, we hope, would help push the innovation frontier on both fronts, the paradigm itself and the applications built atop of it.

[1]The reader is referred to https://bit.ly/2OlGRb6. From [46]: *"In the final print of the series, Picasso reduces the bull to a simple outline which is so carefully considered through the progressive development of each image, that it captures the absolute essence of the creature in as concise an image as possible."*

SIGMOD'20, June 14–19, 2020, Portland, OR, USA
© 2020 Association for Computing Machinery.
ACM ISBN 978-1-4503-6735-6/20/06…$15.00
https://doi.org/10.1145/3318464.3383130

ACM Reference Format:
Anurag Khandelwal, Arun Kejariwal, and Karthik Ramasamy. 2020. *Le Taureau*: Deconstructing the Serverless Landscape & A Look Forward. In *Proceedings of the 2020 ACM SIGMOD International Conference on Management of Data (SIGMOD'20), June 14–19, 2020, Portland, OR, USA.* ACM, New York, NY, USA, 10 pages. https://doi.org/10.1145/3318464.3383130

1 INTRODUCTION

A recent report from *Markets and Markets* reported that the serverless market is expected to reach USD 14.93B by 2023, at a CAGR of 28.6% during the 2017–2023 period [59]. In [152], Passwater reported that the use of serverless has been increasingly growing from fringe to critical workloads. The report also throws light on how serverless is being used, for example: (i) high-availability alerting platform (ii) ETL tool extracting and translating exif data from photos into a heat map (iii) fermentation temperature monitoring with a Raspberry Pi. As per Inkwood research, IT & Telecom is the major industrial vertical in the serverless architecture market [30]. Adoption of serverless has percolated to other industries as well [61, 140]. Looking ahead, real-time streaming processing, IoT applications [83, 105, 155], chatbots/voice-enabled assistants [184], security applications (virus scanning, compliance checking, and incident response [108]) and supercomputing [82] are expected to drive the growth of the serverless market.

Today, several public clouds vendors already support serverless. Examples include AWS Lambda [18], Azure Functions [22], Google Cloud Functions [34] and IBM Cloud Functions [40]. Even in the open source world, several platforms for serverless have emerged, viz., Apache OpenWhisk [13], Fission [51], OpenLambda [107], OpenFaaS [52], SAND [67] and Archipelago [163]. Further, the serverless paradigm is being extended to databases [28], networking and the edge [84, 128, 164, 178]. Use cases include, but not limited to, supporting in-network function execution and enabling reliable communication between Lambda functions.

There exist many sources that overview serverless [60, 115]. Even formal models of serverless have been proposed [98, 113]. However, there does not exist a unified view tying together the serverless paradigm with the applications, platforms and how serverless is being used for analytics and machine learning. To the best of our knowledge, this is the first proposal for a tutorial addressing the aforementioned gap. The rest of the tutorial is organized as follows:

- First, in §2, we shall walk the audience through the evolution of virtualization (early work dates back to the 1960s) and event-driven systems that culminated into serverless. We shall deep dive into *"What is Serverless?"* and it's different flavors.
- In §3, we overview the first generation applications that drove the adoption of serverless.
- Next, in §4, we review the state-of-the-art of serverless platforms. Further, in contrast to the typical use of serverless in the context of stateless computation, we share lessons learned from how Apache Pulsar [14] and Jiffy can be used in tandem to enable stateful computation in the serverless arena.
- In §5, we illustrate how serverless is being (or can be) leveraged for analytics and machine learning.
- Lastly, in §6, we present our views on the opportunities going forward in the serverless landscape.

2 WHAT IS SERVERLESS?

Several recent papers and surveys have attempted to define the concept of serverless computing [60, 81, 106, 107, 115]. While most of these definitions have nuanced differences, the common theme across them mandates the following requirements from serverless platforms:

- *Easy of use*: One of the main goals of serverless computing is to ease the burden on a cloud programmer. As such, serverless platforms must provide simplified access to cloud resources through a combination of high-level programming languages, minimal configuration, and delegation of management issues such as fault-tolerance, load-balancing, system upgrades, etc., to the cloud-provider.
- *Demand-driven execution*: The promise of serverless computing is to enable fine-grained resource elasticity, i.e., the platform should be able to allocate (and de-allocate) resources for an application based on its workload requirements over time. This removes the burden of scaling resources from the user, and allows the cloud provider to efficiently share resources across different users.
- *Cost efficiency*: Finally, the key economic incentive for the users stems from the cost-savings due to fine-grained billing. Combined with fine-grained elasticity, this essentially implies users only pay for the resources they actually use, and for the duration that they use it. This is in contrast to the server-centric model, where the users have to *reserve* server resources regardless of whether or not they use it.

2.1 Evolution of Serverless

The origins of serverless can be traced back to the emergence of *virtualization* [77, 87], i.e., the logical division of resources on physical servers across virtual machines (VMs). While VMs were introduced as a means to share on-premise servers across different user applications, they paved the way to public cloud platforms such as EC2 [4], Azure Compute [20], Google Cloud Platform [33], etc., which used VMs to share massive clusters of servers across multiple tenants. The next step in this evolution was the emergence of *containerization* [25, 27, 167]. Containers *raised* the level of virtualization — while VMs virtualized physical hardware at the operating system-level, containers virtualized the operating system itself, and presented the abstraction of packaged processes as isolated units of execution. In fact, container orchestration services such as Docker [26], Kubernetes [45] and Borg [177], most closely resemble modern serverless compute platforms. The key distinction lies in serverless Function-as-a-Service (FaaS) platforms such as AWS Lambda [18], Azure Functions [22] and Google Cloud Functions [34], raising the virtualization abstraction one step further: they virtualize the runtime, or the process, allowing multiple isolated execution units (typically "functions" or "lambdas") to share the same runtime.

2.2 Serverless Today

Serverless compute platforms were popularized when Amazon launched its AWS Lambda service in 2014 [17]. AWS Lambda was launched under a Function-as-a-Service or FaaS model, where users paid to run stateless *functions* on a platform managed by the provider, without having to deal with the complexity of building and managing complex distributed systems that enable their execution. Many other FaaS platforms have since cropped up, both in industry [3, 13, 22, 34, 40] and in academia [107].

Although the term serverless has become almost synonymous with FaaS platforms today, FaaS platforms are only a part of the serverless ecosystem. The other half of serverless is often referred to as Backend-as-a-Service or BaaS, and comprises cloud-provider managed platforms that enable services beyond stateless compute, including blob storage (e.g., Amazon S3 [8], Azure Blob Storage [43], Google Cloud Storage [36], etc.), database engines (e.g., Amazon's Aurora [69], Google's BigQuery [32], Microsoft's Azure SQL Database serverless [23], etc.), and specialized compute platforms (e.g., Google App Engine [31], AWS Glue [16], etc.). In fact, many of these platforms *predate* FaaS platforms; for instance, Amazon launched S3 in 2006, 8 years before AWS Lambda. The key observation is that while BaaS platforms meet the definitional requirements of serverless platforms as outlined above, they do not aim for generality, i.e., BaaS platforms are often specialized, and serve specific use-cases, as opposed to FaaS platforms that target a more general class of applications.

3 TODAY'S SERVERLESS APPLICATIONS

The benefits of serverless platforms outlined in §2 have lead its widespread adoption across several classical and emerging application domains. In this section, we provide examples of popular serverless applications in use today (§3.1), and characterize them in terms of their workloads (§3.2).

3.1 Examples

We describe three popular serverless use-cases today:

- *Web Applications:* Web applications are perhaps the most common use-case for serverless frameworks [1, 115]. To host such applications [58] in a serverless deployment, the data corresponding to the web content (*e.g.*, HTML, CSS, etc.) and any additional database would be stored on a serverless data store. The processing required in such applications is handled entirely in an event-driven fashion, where some interactive element in the web application leads to a serverless function being executed.
- *Data Processing:* ETL (Extract, Transform, and Load) is another common application for serverless frameworks [16, 57]. The typical use case is to read data from some serverless data store, process it using a serverless function to extract, modify and write useful elements of the data back to serverless storage. The actual extraction and transformation steps may vary, and can be either event-driven or scheduled.
- *Internet of Things:* IoT management is a popular use case for serverless platforms. IoT applications can vary widely based on the device type and the data type they generate [83, 105, 155, 182]. One particular use case is device registration management [42] — whenever a new IoT device registers, it triggers a serverless function, which in turn populates a registry in a serverless data store. The stored registry can then be queried using other serverless functions.

While these examples are representative of serverless, they are certainly not exhaustive. Some of the other applications include integrating third party services, internal tooling, chat-bots (*e.g.*, Alexa Skills) [184], etc., and are outside the scope of this proposal.

3.2 Characteristics

Interestingly, the applications discussed above share several common characteristics outlined below:

- *Need for arbitrary elasticity*: This stems from variable load over time, with the peak load being several times higher than the mean, and the minimum often being zero.
- *Need for cost efficiency*: Despite the variable load, application developers want to only pay for the resources they actually use.
- *Statelessness*: The aforementioned applications *do not* need to share or exchange state between their component execution units (i.e., functions or lambdas).

We note that while these characteristics make these applications an ideal fit for serverless frameworks that exist today, a large class of applications that only satisfy the first two properties are unable to exploit existing serverless frameworks. We refer to such applications as *stateful*, and defer their treatment to §4.4 and §5.

Several prior works have attempted to characterize serverless applications and workloads. Hong et. al. [108] present a categorization of serverless design patterns, viz., (1) periodic invocation, (2) event-driven, (3) data transformation, (4) data streaming, (5) state machine, and (6) bundled pattern. Eyk et al. [176] throw light on critical breakthroughs in six dimensions – what to execute, how to execute, how to find the executable, how to model the program and when to execute – that paved the way to serverless as we know today. Koller and Williams [126] make a case for the end of ubiquity of Linux kernel in the cloud owing to the increasing complexity of the kernel and that the unit of execution on the cloud continues to shrink and development of “native serverless" OS.

4 SERVERLESS PLATFORMS

In this section, we will first describe the various existing serverless platforms, and highlight their high-level characteristics. We will then describe two emerging platforms for serverless ecosystems: *Apache Pulsar*, a pub-sub system that natively supports serverless functions and *Jiffy*, a virtual memory layer for serverless applications.

4.1 FaaS and BaaS

Recall from §2.2 that the serverless ecosystem today is dichotomized into Function-as-a-Service (FaaS) and Backend-as-a-Service (BaaS) platforms.

FaaS. Despite their differences, most FaaS platforms retain several common properties:

- *High-level programming languages*: Most FaaS platforms allow programmers to write their functions using high-level languages such as Python or Javascript.
- *Stateless functions*: These functions are typically stateless, i.e., any state computed during the lifetime of the function is automatically removed when the function execution is over, unless explicitly persisted to an external store.
- *Limited-execution times*: Cloud providers typically limit the execution time of each function to a short duration, often of the order of a few minutes.
- *Fine-grained billing*: Finally, most cloud provider charge customers at fine-grained time units, typically per a few hundred milliseconds of execution.

There are several other nuances in which FaaS platforms differ. These span multiple domains such as security, isolation guarantees, performance, etc. A rigorous analysis of these issues across FaaS platforms can be found in [180].

Recently, programming frameworks such as Ripple [117] have been proposed whereby, applications written for single-machine execution can take advantage of the task parallelism of serverless. Cloudburst [168] is a stateful FaaS platform that provides familiar Python programming with low-latency mutable state and communication.

BaaS. While FaaS platforms provide a general-purpose execution environment for serverless applications, BaaS platforms are typically more specialized, and span varied use-cases. We outline some of the key specializations that exist in BaaS ecosystem today.

- *Storage platforms.* BaaS storage platforms include blob stores such as Amazon S3 [8], Azure Blob Storage [43], Google Cloud Storage [36]. These services allow users to scale their storage to arbitrary sizes. Moreover, the users are billed only for the amount of storage they utilize, and the volume of reads and writes performed on the storage system. Since FaaS platforms are stateless, the storage services provide a means to store state in the serverless ecosystem.
- *Database platforms.* In contrast to the simpler blob-based storage services, serverless database platforms such as AWS Aurora [69] and Azure SQL Database Serverless [23] facilitate structured data storage and richer query semantics. More importantly, since most FaaS platforms re-execute functions transparently on failure, the transactional semantics offered by serverless database services can be crucial for ensuring correctness of serverless functions for several applications.
- *Specialized compute platforms.* Often the stateless nature of FaaS platforms introduces inefficiencies in the execution of common-case application workloads. To cater to such situations, cloud providers have recently introduced a number of specialized serverless compute platforms such as AWS Glue [16] for ETL workloads, Amazon Athena [68], Google BigQuery [32] and Azure Stream Analytics [39] for analytic workloads, etc.

4.2 Orchestration Frameworks

While FaaS platforms are useful for simple stateless applications such as ETL and web-serving (§3), many distributed applications are complex and cannot be expressed in terms of a single function. FaaS orchestration frameworks allow users to compose multiple functions to enable more complex application semantics. Existing orchestration frameworks such as AWS Step Functions [19], IBM Composer [41] and Azure Durable Functions [21] often have varying pricing models, programming models, execution environments, state management solutions, etc. Lopez et. al. [137] provide a comprehensive overview of orchestration frameworks along these dimensions, and outline three key properties that such frameworks should satisfy:

- First, the framework should view each function as a black box, i.e., composing functions should not require the knowledge or modification of their inner workings.
- Second, the composition of several functions defined in the orchestration should also be a function.
- Finally, when running a composition of functions, a user should only be charged for the basic functions, not the composition as well, i.e., they should not be double-billed.

These three properties ensure that orchestration frameworks retain all the benefits of serverless architectures.

FaaS, BaaS, and orchestration frameworks that exist today facilitate the applications outlined in §3. However, recent years have also seen the emergence of new serverless frameworks to make their benefits accessible to a wider range of distributed applications. One such framework is *Apache Pulsar*, which aims to enable serverless streaming applications; we discuss the platform in the following subsection.

4.3 Apache Pulsar

Messaging systems are commonly used in the context of real-time streaming analytics. Apache Pulsar is an enterprise-grade messaging system that was originally developed at Yahoo [14].[2] Pulsar generalizes the traditional messaging models — queuing and publish-subscribe ("pub-sub") — through one unified messaging API. Some of the other key features of Pulsar include support for geo-replication, multi-tenancy, tiered storage and failure recovery. In Pulsar, producers publish messages to topics and consumers subscribe to one or more topics to ingest messages. Pulsar is designed to operate at any scale — data fed into a topic can range from a few megabytes to several terabytes. Pulsar supports partitioned topics in order to scale to large data volumes, via a process called a broker; each node in a Pulsar cluster runs its own broker (see Figure 1 for a system overview).

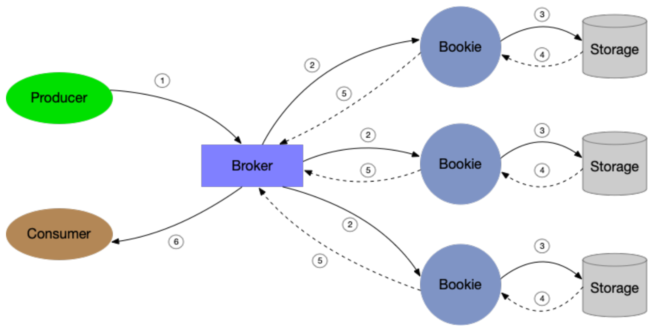

Figure 1: System Overview

A Pulsar cluster is composed of a set of brokers and bookies and an Apache ZooKeeper [110] ensemble for coordination and configuration management. The Pulsar broker is the component that receives, stores and delivers messages. The bookies are from Apache BookKeeper [10] that provide durable stream storage for messages until they are consumed.

Broker. The Pulsar broker is a stateless component and is tasked with receiving and dispatching messages while using

[2]For comparison of Pulsar with other popular messaging systems such as Apache Kafka, Amazon Kinesis Streams and RabbitMQ, see [63–65].

bookie as durable storage for messages until they are consumed. The broker primarily runs two different components: a HTTP server that exposes a REST interface for topic lookup and administrative tasks, and a native dispatcher, which is an asynchronous TCP server over a custom binary protocol used for all data transfers.

Bookie. Pulsar's storage nodes are called bookies, and are based on Apache Bookkeeper, a distributed write-ahead log system that provides the durable storage. Applications using Bookkeeper can create many independent logs, called ledgers. A ledger is an append-only data structure with a single writer that is assigned to multiple bookies, and their entries are replicated to multiple bookie nodes. The semantics of a ledger are very simple: a process can create a ledger, append entries and close the ledger. After the ledger has been closed, either explicitly or because the writer process crashed, it can only be opened in read-only mode. Finally, when the entries contained in the ledger are no longer needed, the whole ledger can be deleted from the system.

4.3.1 Pulsar Functions. Pulsar functions allow users to deploy and manage processing of serverless functions (see §4.1) that consume messages from and publish messages to Pulsar topics [53]. Pulsar functions serves a means to support analytics on real-time data streams in a serverless fashion. This is exemplified later in §5.1. Many data analytics algorithms are stateful in nature. This would limit the use of today's serverless frameworks as they were designed for stateless applications. Likewise, composable and/or concurrent data sketches [157] – these require passing ephemeral data between different sketches – cannot be supported in today's serverless frameworks. To alleviate this, recently, systems such as *Jiffy* have been proposed for storage and management of ephemeral state. We discuss the same in the next subsection.

4.4 Jiffy

A necessary component in the execution of many distributed applications is data sharing exchange across their component tasks (or functions). Such data is referred to as *ephemeral state*, due to its short-lived nature. Examples of such applications range from data flow programs [111], graph analytics [141], video processing [97], machine learning and AI [48, 147] to stream processing [136, 185].

Unfortunately, as we saw in §3, serverless frameworks today mainly cater to *stateless* applications, and do not facilitate efficient storage and exchange of ephemeral application state. Three unique properties of serverless architectures introduce new challenges to achieving this goal, that render existing storage systems ineffective for stateful applications:

- **No support for direct communication:** Since applications have no control over placement and scheduling of individual tasks, inter-task state exchange must resort to external stores instead of using direct communications. Existing persistent stores [8, 36, 43] unfortunately do not provide the required performance for such exchange.
- **Elasticity and isolation are first-class citizens:** In keeping with the promise of serverless, ephemeral state storage needs to provide transparent scaling of resources, while providing *isolation*, i.e., scaling up/down the ephemeral storage resources for one application should not impact other concurrent applications [124]. Existing approaches either only provide coarse-grained resource elasticity with no isolation [7, 47, 92, 151], or require knowing the number of concurrent tasks and ephemeral state size in advance [125, 156], which is infeasible for most serverless applications.
- **Lifetime management is challenging:** Existing serverless platforms tightly couple the lifetime of state with that of its producer task. However, in most applications, lifetime of shared state may be much longer than that of the producer task: it is tied to when data is *consumed.* Unfortunately, data stores today do not provide state lifetime management [7, 47, 92, 151, 165].

In this tutorial, we will describe Jiffy, a virtual memory system that enables storage and management of ephemeral state for serverless applications. Jiffy builds on three core insights:

- While it is hard to accurately provision capacity for any *individual* application, it is possible to exploit the short-lived nature of serverless tasks to efficiently multiplex the available memory capacity *across* applications.
- A single global address space, as exposed in classical distributed shared memory systems [80, 89, 96, 118, 133] and recent in-memory stores [54, 92, 135, 151], precludes isolation guarantees for scaling memory resources in multi-tenant settings, since adding/removing memory resources for an application requires re-partitioning data for the entire address-space. Such settings necessitate a design that breaks the single global address-space abstraction.
- Fine-grained memory elasticity and sharing of memory resources with isolation are precisely the problems that traditional operating systems have already solved with virtual memory abstractions. Resolving them in the serverless setting should build on the design insights in these systems.

Jiffy incorporates these insights into a system with two complementary approaches: (1) block-level memory allocation across a shared pool of memory nodes (akin to page-level allocations in operating systems), and (2) a serverless centric memory management abstraction — *hierarchical namespaces* (akin to virtual address-spaces). Hierarchical namespaces, with sub-namespaces for sub-tasks, allows capturing the ephemeral state dependency between an application's tasks. Capacity allocation and de-allocation at the granularity of blocks allows Jiffy to efficiently multiplex the available memory capacity across applications. Moreover, adding/removing

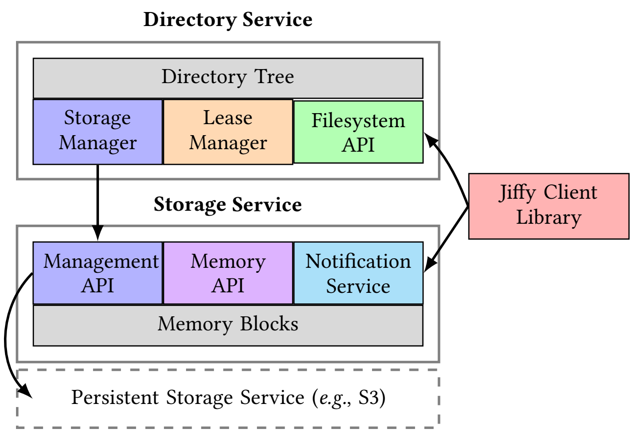

Figure 2: Jiffy Design Overview

blocks to a task's sub-namespace requires re-partitioning the data *only* for that sub-namespace, ensuring both efficiency and isolation for elastic scaling of memory resources. Finally, namespaces naturally enable lifetime management using a namespace-granularity leasing mechanism [103], and signaling to applications when relevant state is ready for processing using a per-namespace notification mechanism [9, 56]. Figure 2 shows the high-level architecture for Jiffy.

5 SERVERLESS FOR DATA INSIGHTS

Massively parallel applications – be it the traditional Monte Carlo simulation or the contemporary hyperparameter tuning for machine learning [93, 116, 134, 161, 166] – lend themselves naturally to the serverless paradigm. More recently, driven by time-to-market constraints, there has been a large amount of work to speedup model training [70, 90], deep reinforcement learning [149] – this will help to boost the development on next-generation AI applications. Further, via divide-and-conquer, parallelism at different granularities is being surfaced and exploited via serverless. In this section, we present a snapshot of how serverless is being leveraged for applications which are ubiquitous today and are expected to grow materially going forward.

5.1 Analytics

We now present a flavor of the various use cases of serverless for analytics in a wide variety of application domains.

- **Video**: As per Cisco's VNI report from 2019, globally, IP video traffic will be 82% of all IP traffic by 2022 [24]. Live video will account for 17% of Internet video traffic by 2022 (a 15× growth from 2017). In light of the above, there has been emergence of new applications rooted in a wide spectrum of video analysis – video editing, scene understanding, object recognition and classification, and compositing. Leveraging serverless enables the scaling needed given the compute intensive and low latency nature of the applications.

  We describe two serverless frameworks for video processing. ExCamera [97] is a video processing framework that facilitates fine-grained parallelism for video encoding on AWS Lambda. Sprocket is a serverless video processing framework that exploits intra-video parallelism to achieve low latency [71]. Zhang et al. [187] presented a measurement study of video processing with serverless computing.
- **Graph Processing**: In application domains such as social networks, traffic engineering, bioinformatics, telecommunications graph processing plays a central role in surfacing actionable insights. To this end, several frameworks have been developed over the years [12, 172, 174].

  Recently, Toader et al. [173] presented a serverless approach to graph processing. It employs the Pregel computation model [142] as its execution model and uses a memory engine – which serves as an abstraction over a distributed set of Redis instances [55] – to store intermediate state during graph processing.
- **Sequence comparison**: Determining similarity between protein sequences is often a prerequisite to clustering sequences into related families or functional groups. Niu et al. [150] illustrate the use of serverless to carry out an all-to-all pairwise comparison among all unique human proteins.

  Other tasks from bioinformatics amenable to serverless [129] include, for example, sequence alignment [109], protein folding, visualization of DNA sequences [131].
- **Matrix Multiplication**: As Deep Learning has become ubiquitous across the various modalities – viz., text, audio, images and video – the importance of `MATVEC` and `MATMTUL`[3] has grown beyond HPC applications. Distributed execution of the above requires support for ephemeral storage of intermediate results (refer to §4.4). Werner et al. [181] illustrated distributed execution of Strassen's algorithm [170] for `MATMTUL` in a serverless setting.
- **Extract, Transform, Load (ETL)**: A recent report from InForGrowth forecasted a growth of over 10% of the global ETL market until 2025 [62]. Fingler et al. [95] presented an unikernel[4]-based design, called USETL, specialized for serverless ETL workloads [139]. Given that ETL workloads have limited interaction with networking and storage, the authors proposed to virtualize I/O at the runtime library interface to reduce the memory and CPU overhead.

Besides the use cases discussed above, there's a rich family of data sketches – *sampling, filtering, quantiles, cardinality, frequent elements, clustering, moments, wavelets, graph sketches, matrix sketching* – that can benefit from the properties of serverless discussed earlier in §2. Figure 3 illustrates how the data sketch `Count-Min` (to estimate event frequency) [86] can be implemented as a Pulsar function (§4.3).

[3]The two kernels form the basis of how most of today's deep learning algorithms are executed.
[4]Unikernels comprise of a minimal operating system kernel and a single application baked with it, in a single address space [139].

```
import org.apache.pulsar.functions.api.Context;
import org.apache.pulsar.functions.api.Function;
import com.clearspring.analytics.stream.frequency.CountMinSketch;

public class CountMinFunction implements Function<String, Void> {
   CountMinSketch sketch = new CountMinSketch(20,20,128);

   Void process(String input, Context context) throws Exception {
      sketch.add(input, 1); // Calculates bit indexes and performs +1
      long count = sketch.estimateCount(input);
      // React to the updated count
      return null;
   }
}
```

Figure 3: `Count-Min` sketch as a Pulsar function

The reader is referred to [123] for other examples on how other commonly used data sketches can be implemented using Pulsar Functions. Further, sketching can be use to augment the speed of training of ML models [146]. An more comprehensive overview of prior work on algorithms for mining insights from streaming data can be found in [66, 85, 102, 144, 148, 153, 162], and in the tutorial by Kejariwal et al. [120]. Open-source software on data sketches [2, 11] can potentially also be adapted to the serverless context.

In recent years, several frameworks for serverless data analytics have been proposed [15, 114, 122, 158]. PyWren [114] is a serverless data analytics engine that runs on AWS Lambda. Sampe et al. [158] presented an extension of PyWren to run broader MapReduce jobs. García-López et al. [100] provide a discussion of the tradeoffs and challenges of serverless data analytics.

5.2 Machine Learning

Today, machine learning (ML)[5] is ubiquitous in almost every product. Driven by the need to deeply personalize the end-user experience and/or surfacing highly actionable insights, ML models have grown in complexity and capacity [74, 119, 171, 179]; likewise, the data volume to train these models has also grown significantly. Fast model training is critical for being able to iterate rapidly; in a similar vein, fast model inference is key for good user experience – be it latency or battery life (the latter comes into play in the context of federated learning [76, 127, 145], wherein a ML model is run on an user's device). To this end, serverless is increasingly being used [79]. In this subsection, we dive into how serverless is being used for ML workloads.

- **Training**: Serverless serves as a natural way to exploit data parallelism during model training. Specifically, a dataset is partitioned into multiple subsets and then each subset is used to train a given model in parallel on independent serverless instances. Gradients computed by all the instances are collected by a parameter server, which then updates the network parameters. Feng et al. [94] proposed a hierarchical update and reuse of parameter servers to minimize the latency associated with spinning up a new instances. Zhang et al. [186] presented a system for training as well as for hyperparameter tuning. In particular, the system concurrently invokes functions for all combinations of the hyperparameters specified and returns the configuration that results in the best score.

  Model training is typically cast as an optimization problem. Commonly used algorithms in this regard are iterative in nature and hence, are stateful. In light of this, Aytekin and Johansson proposed a setup of master-worker nodes (along with a scheduler)for serverless model training [73]. In a similar vein, Gupta et al. proposed a randomized Hessian-based optimization algorithm to solve large-scale convex optimization problems in serverless systems [104]. The algorithm supports in-built resiliency against stragglers that are characteristic of serverless architectures. This is achieved based on error-correcting codes to create redundant computation [132].

  Given that many tasks in model training are stateful, use of ephemeral storage such as Jiffy (§4.4) can help drive further adoption of serverless for model training.
- **Inference**: Fast inference is critical in today's ML-driven applications, e.g., driving buy/sell decisions in stock trading [183], real-time bidding in the context of advertising [78, 188], real-time speech synthesis [72, 91, 154], anomaly detection [189], live video analysis [99, 138], etc.[6]

  Ishakian et al. empirically showed that that warm serverless executions are within an acceptable latency range, while cold starts add significant overhead [112]. Dakkak et al. [88] proposed an approach, comprising of a persistent model store across the GPU, CPU, local storage, and cloud storage hierarchy and a resource management layer that provide isolation, to address the cold start latency issue. Bhattacharjee et al. [75] presented a system to minimize the total cost incurred while ensuring bounded prediction latency (via horizontal and vertical scaling) with reasonable accuracy. The system has in-built support to forecast changes in resource demand (this may be induce, for instance, by a sudden increase in incoming traffic) and make effective and pro-active resource allocation decisions.

We conclude in the next section with a brief discussion of the challenges and opportunities ahead.

6 CONCLUSIONS AND LOOKING AHEAD

We conclude this tutorial with a description of the missing pieces in serverless infrastructures today, and discuss ways in which future research and development can address them.

Security. Existing serverless platforms are susceptible to security vulnerabilities via side-channels due to increased co-residency [115]. Moreover, FaaS platforms lead to increased network communications due to external storage accesses [8, 43], leaking more information to a network adversary. As another example, functions of different tenants may run on

[5]In the rest of the proposal, ML is used to additionally represent deep learning and reinforcement learning.

[6]Note that inference is computationally less intensive than training; hence, it is more sensitive to data loading and deserialization overhead.

the same physical hardware, increasing the likelihood of traditional side-channel attacks like Rowhammer [159].

Recent research has focused on lightweight isolation between functions on shared hardware via secure containers [29, 38, 44, 49, 143]. This may potentially be extended to isolation at the *hardware* layer. Increased network communications incentivizes the exploration of security primitives that hide network access patterns in the cloud, *e.g.,* using ORAMs [101, 169] or anonymous communications [130, 175].

SLA Guarantees. Higher resource sharing also leads to decreased performance predictability in serverless frameworks: the variability stems, in part, from allocation and scheduling delays, cold start latencies, etc.

Providing performance isolation can also be achieved at lower layers of the serverless stack; *e.g.*, in container design [121, 160], and at the hardware level. Future research may explore *bin-packing* techniques that "pack" different functions together based on heuristics that ensure performance isolation, *e.g.*, by packing together functions that have complementary performance goals (*e.g.*, throughput/latency) or resource requirements (*e.g.*, CPU/GPU/TPU), ensuring they do not contend with each other.

Hardware Heterogeneity. Existing platforms mainly cater to users with general-purpose compute needs, but largely ignore users that rely on specialized compute resources like GPUs, TPUs and FPGAs. While cloud providers offer VMs that have access to GPUs [6, 35, 50], FPGAs [5] and TPUs [37], serverless platforms are yet to adopt them. However, the lack of these resources in the serverless ecosystem is not fundamental; we believe that growing user demand will drive cloud providers to offer them over time.

Serverless is the Future. Despite the missing pieces outlined above, the general consensus in the community is that serverless is the future of the cloud [106, 115]. The key driving factor is that serverless frameworks provide strong incentives for all the players involved: (1) users benefit from the cost-savings due to fine-grained billing; (2) the cloud provider benefits due to the cost-savings arising from higher degree of resource multiplexing and increased resource utilization; and, (3) the developers benefit perhaps the most from the ease of programming. We believe that these strong incentives will result in cloud providers addressing most, if not all, of the issues outlined above in the near future.

REFERENCES

[1] [n.d.]. 2018 Serverless Community Survey: huge growth in serverless usage. https://bit.ly/2Mu5TCR.
[2] [n.d.]. Algebird. ([n. d.]). https://twitter.github.io/algebird/datatypes/approx.html.
[3] [n.d.]. Alibaba Functions Compute. "https://www.alibabacloud.com/products/function-compute".
[4] [n.d.]. Amazon EC2. "https://aws.amazon.com/ec2/".
[5] [n.d.]. Amazon EC2 F1 Instances. "https://aws.amazon.com/ec2/instance-types/f1/".
[6] [n.d.]. Amazon EC2 P2 Instances. "https://aws.amazon.com/ec2/instance-types/p2/".
[7] [n.d.]. Amazon ElastiCache. "https://aws.amazon.com/elasticache".
[8] [n.d.]. Amazon S3. "https://aws.amazon.com/s3".
[9] [n.d.]. Amazon Simple Notification Service (SNS). https://aws.amazon.com/sns.
[10] [n.d.]. Apache BookKeeper: A scalable, fault-tolerant, and low-latency storage service optimized for real-time workloads. ([n. d.]). http://bookkeeper.apache.org/.
[11] [n.d.]. Apache DataSketches. ([n. d.]). http://datasketches.apache.org/.
[12] [n.d.]. Apache Giraph. ([n. d.]). https://giraph.apache.org/.
[13] [n.d.]. Apache OpenWhisk. "https://openwhisk.apache.org".
[14] [n.d.]. Apache Pulsar. ([n. d.]). https://pulsar.apache.org/.
[15] [n.d.]. Automate the Data Science Pipeline with Serverless Functions. ([n. d.]). https://nuclio.io.
[16] [n.d.]. AWS Glue. "https://aws.amazon.com/glue/".
[17] [n.d.]. AWS Lambda – Run Code in the Cloud. "https://aws.amazon.com/blogs/aws/run-code-cloud/".
[18] [n.d.]. AWS Lamda. "https://aws.amazon.com/lambda/".
[19] [n.d.]. AWS Step Functions. "https://aws.amazon.com/step-functions/".
[20] [n.d.]. Azure Compute. https://azure.microsoft.com/en-us/product-categories/compute/.
[21] [n.d.]. Azure Durable Orchestrations. "https://docs.microsoft.com/en-us/azure/azure-functions/durable/durable-functions-orchestrations".
[22] [n.d.]. Azure Functions. "https://azure.microsoft.com/en-us/services/functions".
[23] [n.d.]. Azure SQL Database Serverless. "https://docs.microsoft.com/en-us/azure/sql-database/sql-database-serverless".
[24] [n.d.]. Cisco Visual Networking Index: Forecast and Trends, 2017–2022 White Paper. ([n. d.]). https://www.cisco.com/c/en/us/solutions/collateral/service-provider/visual-networking-index-vni/white-paper-c11-741490.html.
[25] [n.d.]. Consolidating Applications with Oracle Solaris Containers. "http://www.oracle.com/us/products/servers-storage/solaris/consolid-solaris-containers-wp-075578.pdf".
[26] [n.d.]. Docker. "https://www.docker.com/".
[27] [n.d.]. Docker: Lightweight Linux Containers for Consistent Development and Deployment. "https://www.linuxjournal.com/content/docker-lightweight-linux-containers-consistent-development-and-deployment".
[28] [n.d.]. Fauna: The database built for serverless, featuring native GraphQL. ([n. d.]). https://fauna.com/.
[29] [n.d.]. Firecracker – Lightweight Virtualization for Serverless Computing. "https://aws.amazon.com/blogs/aws/firecracker-lightweight-virtualization-for-serverless-computing/".
[30] [n.d.]. Global Serverless Architecture Market Forecast 2019-2027. ([n. d.]). https://www.inkwoodresearch.com/reports/global-serverless-architecture-market/.
[31] [n.d.]. Google App Engine. "https://cloud.google.com/appengine/".
[32] [n.d.]. Google BigQuery. "https://cloud.google.com/bigquery/".
[33] [n.d.]. Google Cloud. https://cloud.google.com/.
[34] [n.d.]. Google Cloud Functions. "https://cloud.google.com/functions".
[35] [n.d.]. Google Cloud GPUs. "https://cloud.google.com/gpu/".
[36] [n.d.]. Google Cloud Storage. "https://cloud.google.com/storage/".
[37] [n.d.]. Google Cloud TPU. "https://cloud.google.com/tpu/".
[38] [n.d.]. gVisor: A container sandbox runtime focused on security, efficiency, and ease of use. "https://gvisor.dev/".
[39] [n.d.]. https://azure.microsoft.com/en-us/services/stream-analytics/. https://azure.microsoft.com/en-us/services/stream-analytics/.
[40] [n.d.]. IBM Cloud Functions. "https://www.ibm.com/cloud/functions".
[41] [n.d.]. IBM Composer. "https://cloud.ibm.com/docs/openwhisk?topic=cloud-functions-pkg_composer".
[42] [n.d.]. Implementing a Serverless AWS IoT Backend with AWS Lambda and Amazon DynamoDB. "https://aws.amazon.com/serverless/build-a-web-app/".
[43] [n.d.]. Introduction to object storage in Azure. https://docs.microsoft.com/en-us/azure/storage/blobs/storage-blobs-introduction.
[44] [n.d.]. Kata containers: The speed of containers, the security of VMs. "https://katacontainers.io/".
[45] [n.d.]. Kubernetes. "https://kubernetes.io/".
[46] [n.d.]. Le Taureau, by P. Picasso, 1946. ([n. d.]). https://www.wikiart.org/en/pablo-picasso/bull-plate-xi-1946.
[47] [n.d.]. MemCached. http://www.memcached.org.
[48] [n.d.]. MLlib. https://spark.apache.org/docs/latest/ml-guide.html.
[49] [n.d.]. Nabla Containers: A new approach to Container Isolation. "https://nabla-containers.github.io/".
[50] [n.d.]. NVv4 Azure Virtual Machines for GPU visualization workloads. "https://azure.microsoft.com/en-us/blog/introducing-nvv4-azure-virtual-machines-for-gpu-visualization-workloads/".
[51] [n.d.]. Open source, Kubernetes-native Serverless Framework. ([n. d.]). https://fission.io/.
[52] [n.d.]. OpenFaaS. ([n. d.]). https://github.com/openfaas/faas.
[53] [n.d.]. Pulsar Functions overview. ([n. d.]). https://pulsar.apache.org/docs/v2.0.1-incubating/functions/overview/.
[54] [n.d.]. Redis. http://www.redis.io.
[55] [n.d.]. Redis Cluster Specification. https://redis.io/topics/cluster-spec.
[56] [n.d.]. Redis Keyspace Notifications. https://redis.io/topics/notifications.
[57] [n.d.]. Serverless ETL using Lambda and SQS. "https://medium.com/poka-techblog/serverless-etl-using-lambda-and-sqs-d8b4de1d1c43".
[58] [n.d.]. Serverless Web Application. "https://aws.amazon.com/blogs/compute/implementing-a-serverless-aws-iot-backend-with-aws-lambda-and-amazon-dynamodb/".
[59] 2018. Serverless Architecture Market by Service type, Deployment Model, Organization Size, Verticals, and Region - Global Forecast to 2023. (Aug. 2018). https://www.marketsandmarkets.com/Market-Reports/serverless-architecture-market-64917099.html.
[60] 2018. Serverless Architectures. (May 2018). https://martinfowler.com/articles/serverless.html.
[61] 2018. The challenges of developing event-driven serverless systems at massive scale. (2018). https://read.acloud.guru/designing-an-event-driven-serverless-system-to-run-real-time-at-massive-scale-c4de3f7539fc.
[62] 2019. Global ETL Software Market Size, Status and Forecast 2019-2025. (Aug. 2019). https://inforgrowth.com/report/3606853/etl-software-market.
[63] 2019. Life Beyond Kafka With Apache Pulsar. (Oct. 2019). https://dzone.com/articles/life-beyond-kafka-with-apache-pulsar.
[64] 2019. Performance Comparison between Apache Pulsar and Kafka: Latency. (Aug. 2019). https://kafkaesque.io/performance-comparison-between-apache-pulsar-and-kafka-latency/.
[65] 2019. Why We Moved to Apache Pulsar. (2019). https://corp.narvar.com/blog/why-we-moved-to-apache-pulsar/.
[66] C. C. Aggarwal. 2006. *Data Streams: Models and Algorithms (Advances in Database Systems)*. Springer-Verlag.
[67] I. E. Akkus, R. Chen, I. Rimac, M. Stein, K. Satzke, A. Beck, P. Aditya, and V. Hilt. 2018. SAND: Towards High-Performance Serverless Computing. In *Proceedings of the USENIX Annual Technical Conference*. Boston, MA, 923–935.
[68] Amazon. [n.d.]. Amazon Athena. "https://aws.amazon.com/athena".

[69] Amazon. [n.d.]. Amazon Aurora Serverless. "https://aws.amazon.com/rds/aurora/serverless".

[70] R. Anil, G. Pereyra, A. Passos, R. Ormandi, G. E. Dahl, and G. E. Hinton. 2018. Large scale distributed neural network training through online distillation. *The Computing Research Repository* abs/1804.03235 (2018).

[71] L. Ao, L. Izhikevich, G. M. Voelker, and G. Porter. 2018. Sprocket: A Serverless Video Processing Framework. In *Proceedings of the ACM Symposium on Cloud Computing*. Carlsbad, CA, 263–274.

[72] S. Ö. Arik, M. Chrzanowski, A. Coates, G. F. Diamos, A. Gibiansky, Y. Kang, X. Li, J. Miller, A. Ng, J. Raiman, S. Sengupta, and M. Shoeybi. 2017. Deep Voice: Real-time Neural Text-to-Speech. In *Proceedings of the International Conference on Machine Learning*.

[73] A. Aytekin and M. Johansson. 2019. Exploiting Serverless Runtimes for Large-Scale Optimization. In *Proceedings of the IEEE 12th International Conference on Cloud Computing*. 499–501.

[74] P. Baldi and R. Vershynin. 2018. On Neuronal Capacity. In *Proceedings of the Conference on Neural Information Processing Systems*.

[75] A. Bhattacharjee, A. D. Chhokra, Z. Kang, H. Sun, A. Gokhale, and G. Karsai. 2019. BARISTA: Efficient and Scalable Serverless Serving System for Deep Learning Prediction Services. *The Computing Research Repository* abs/1904.01576 (2019).

[76] K. Bonawitz, H. Eichner, W. Grieskamp, D. Huba, A. Ingerman, V. Ivanov, C. Kiddon, J. Konečný, S. Mazzocchi, H. B. McMahan, T. V. Overveldt, D. Petrou, D. Ramage, and J. Roselander. 2019. Towards Federated Learning at Scale: System Design. *The Computing Research Repository* abs/1902.01046 (2019).

[77] E. Bugnion, S. Devine, and M. Rosenblum. 1997. Disco: Running Commodity Operating Systems on Scalable Multiprocessors. In *Proceedings of the Sixteenth ACM Symposium on Operating Systems Principles*. Saint Malo, France, 143–156.

[78] H. Cai, K. Ren, W. Zhang, K. Malialis, J. Wang, Y. Yu, and D. Guo. 2017. Real-Time Bidding by Reinforcement Learning in Display Advertising. In *Proceedings of the Tenth ACM International Conference on Web Search and Data Mining*. Cambridge, UK, 661–670.

[79] J. Carreira, P. Fonseca, A. Tumanov, A. Zhang, and R. Katz. 2019. Cirrus: A Serverless Framework for End-to-End ML Workflows. In *Proceedings of the ACM Symposium on Cloud Computing*. Santa Cruz, CA, 13–24.

[80] J. B. Carter, D. Khandekar, and L. Kamb. 1995. Distributed shared memory: Where we are and where we should be headed. In *Proceedings 5th Workshop on Hot Topics in Operating Systems (HotOS-V)*. 119–122.

[81] P. Castro, V. Ishakian, V. Muthusamy, and A. Slominski. 2019. The Rise of Serverless Computing. *Commun. ACM* 62, 12 (Nov. 2019), 44–54.

[82] R. Chard, T. J. Skluzacek, Z. Li, Y. Babuji, A. Woodard, B. Blaiszik, S. Tuecke, I. Foster, and K. Chard. 2019. Serverless Supercomputing: High Performance Function as a Service for Science. *The Computing Research Repository* abs/1908.04907 (2019).

[83] B. Cheng, J. Fürst, G. Solmaz, and T. Sanada. 2019. Fog Function: Serverless Fog Computing for Data Intensive IoT Services. *The Computing Research Repository* abs/1907.08278 (2019).

[84] S. Choi, M. Shahbaz, B. Prabhakar, and M. Rosenblum. 2019. λ-NIC: Interactive Serverless Compute on Programmable SmartNICs. *The Computing Research Repository* abs/1909.11958 (2019).

[85] G. Cormode. 2007. References for Data Stream Algorithms. (2007). http://dimacs.rutgers.edu/~graham/pubs/papers/bristol.pdf.

[86] G. Cormode and S. Muthukrishnan. 2005. An Improved Data Stream Summary: The Count-min Sketch and Its Applications. *Journal of Algorithms* 55, 1 (April 2005), 58–75.

[87] R. J. Creasy. 1981. The origin of the VM/370 time-sharing system. *IBM Journal of Research and Development* 25, 5 (1981), 483–490.

[88] A. Dakkak, C. Li, S. G. de Gonzalo, J. Xiong, and W. mei Hwu. 2018. TrIMS: Transparent and Isolated Model Sharing for Low Latency Deep LearningInference in Function as a Service Environments. *The Computing Research Repository* abs/1811.09732 (2018).

[89] P. Dasgupta, R. J LeBlanc, M. Ahamad, and U. Ramachandran. 1991. The Clouds distributed operating system. *Computer* 24, 11 (1991), 34–44.

[90] J. Dean, G. S. Corrado, R. Monga, K. Chen, M. Devin, Q. V. Le, M. Z. Mao, M. Ranzato, A. W. Senior, P. A. Tucker, K. Yang, and A. Y. Ng. 2012. Large Scale Distributed Deep Networks. In *Proceedings of the COnference on Neural Information Processing Systems*.

[91] S. Ö. Arik G. F. Diamos, A. Gibiansky, J. Miller, K. Peng, W. Ping, J. Raiman, and Y. Zhou. 2017. Deep Voice 2: Multi-Speaker Neural Text-to-Speech. In *Proceedings of the Conference on Neural Information Processing Systems*.

[92] A. Dragojević, D. Narayanan, O. Hodson, and M. Castro. [n.d.]. FaRM: Fast Remote Memory. In *Proceedings of the Symposium on Networked Systems Design and Implementation*.

[93] S. Falkner, A. Klein, and F. Hutter. 2018. BOHB: Robust and Efficient Hyperparameter Optimization at Scale. *The Computing Research Repository* abs/1810.05934 (2018).

[94] L. Feng, P. Kudva, D. Da Silva, and J. Hu. 2018. Exploring Serverless Computing for Neural Network Training. In *Proceedings of the IEEE 11th International Conference on Cloud Computing*. 334–341.

[95] H. Fingler, A. Akshintala, and C. J. Rossbach. 2019. USETL: Unikernels for Serverless Extract Transform and Load Why Should You Settle for Less?. In *Proceedings of the 10th ACM SIGOPS Asia-Pacific Workshop on Systems*. Hangzhou, China, 23–30.

[96] Brett Fleisch and Gerald Popek. 1989. *Mirage: A coherent distributed shared memory design.* Vol. 23. ACM.

[97] S. Fouladi, R. S. Wahby, B. Shacklett, K. V. Balasubramaniam, W. Zeng, R. Bhalerao, A. Sivaraman, G. Porter, and K. Winstein. [n.d.]. Encoding, Fast and Slow: Low-Latency Video Processing Using Thousands of Tiny Threads. In *USENIX Symposium on Networked Systems Design and Implementation*. 363–376.

[98] M. Gabbrielli, S. Giallorenzo, I. Lanese, F. Montesi, M. Peressotti, and S. P. Zingaro. 2019. No more, no less - A formal model for serverless computing. *The Computing Research Repository* abs/1903.07962 (2019).

[99] O. Gafni, L. Wolf, and Y. Taigman. 2019. Live Face De-Identification in Video. *The Computing Research Repository* abs/1911.08348 (2019).

[100] P. García-López, M. Sánchez-Artigas, S. Shillaker, P. Pietzuch, D. Breitgand, G. Vernik, P. Sutra, T. Tarrant, and A. J. Ferrer. 2019. ServerMix: Tradeoffs and Challenges of Serverless Data Analytics.

[101] Oded Goldreich and Rafail Ostrovsky. 1996. Software protection and simulation on oblivious RAMs. *JACM* (1996).

[102] H. M. Gomes, J. P. Barddal, F. Enembreck, and A. Bifet. 2017. A Survey on Ensemble Learning for Data Stream Classification. *Comput. Surveys* 50, 2, Article 23 (March 2017), 23:1–23:36 pages.

[103] Cary Gray and David Cheriton. 1989. *Leases: An efficient fault-tolerant mechanism for distributed file cache consistency*. Vol. 23. ACM.

[104] V. Gupta, S. Kadhe, T. Courtade, M. W. Mahoney, and K. Ramchandran. 2019. OverSketched Newton: Fast Convex Optimization for Serverless Systems. In *Proceedings of the Thirty-sixth International Conference on Machine Learning*.

[105] A. Hall and U. Ramachandran. 2019. An Execution Model for Serverless Functions at the Edge. In *Proceedings of the International Conference on Internet of Things Design and Implementation*. 225–236.

[106] J. M. Hellerstein, J. Faleiro, J. E. Gonzalez, J. Schleier-Smith, V. Sreekanti, A. Tumanov, and C. Wu. 2018. Serverless computing: One step forward, two steps back. *arXiv:1812.03651* (2018).

[107] S. Hendrickson, S. Sturdevant, T. Harter, V. Venkataramani, A. C Arpaci-Dusseau, and R. H. Arpaci-Dusseau. 2016. Serverless computation with OpenLambda. In *Proceedings of the 8th USENIX Workshop on Hot Topics in Cloud Computing*. Denver, CO. https://github.com/open-lambda/open-lambda.

[108] S. Hong, A. Srivastava, W. Shambrook, and T. Dumitras. 2018. Go Serverless: Securing Cloud via Serverless Design Patterns. In *10th USENIX Workshop on Hot Topics in Cloud Computing (HotCloud 18)*. Boston, MA.

[109] L.-H. Hung, D. Kumanov, X. Niu, W. Lloyd, and K. Y. Yeung. 2019. Rapid RNA sequencing data analysis using serverless computing. *bioRxiv* (2019).

[110] Patrick Hunt, Mahadev Konar, Flavio Paiva Junqueira, and Benjamin Reed. 2010. ZooKeeper: Wait-free Coordination for Internet-scale Systems. In *USENIX Technical Conference (ATC)*.

[111] M. Isard, M. Budiu, Y. Yu, A. Birrell, and D. Fetterly. 2007. Dryad: distributed data-parallel programs from sequential building blocks. In *ACM SIGOPS operating systems review*, Vol. 41. 59–72.

[112] V. Ishakian, V. Muthusamy, and A. Slominski. 2017. Serving deep learning models in a serverless platform. *The Computing Research Repository* abs/1710.08460 (2017).

[113] A. Jangda, D. Pinckney, Y. Brun, and A. Guha. 2019. Formal Foundations of Serverless Computing. *Proceedings of the SIGPLAN Conference on Object Oriented Programming, Systems, Languages, and Applications* 3, Article 149 (Oct. 2019), 149:1–149:26 pages.

[114] E. Jonas, Q. Pu, S. Venkataraman, I. Stoica, and B. Recht. 2017. Occupy the cloud: distributed computing for the 99%. In *Proceedings of the Symposium on Cloud Computing*. 445–451.

[115] E. Jonas, J. Schleier-Smith, V. Sreekanti, C.-C. Tsai, A. Khandelwal, Q. Pu, V. Shankar, J. M. Carreira, K. Krauth, N. Yadwadkar, J. Gonzalez, R. A. Popa, I. Stoica, and D. A. Patterson. 2019. *Cloud Programming Simplified: A Berkeley View on Serverless Computing*. Technical Report UCB/EECS-2019-3. EECS Department, University of California, Berkeley.

[116] T. T. Joy, S. Rana, S. Gupta, and S. Venkatesh. 2019. Fast Hyperparameter Tuning using Bayesian Optimization with Directional Derivatives. *The Computing Research Repository* 1902.02416 (2019).

[117] S. Joyner, M. MacCoss, C. Delimitrou, and H. Weatherspoon. 2020. Ripple: A Practical Declarative Programming Framework for Serverless Compute. *The Computing Research Repository* abs/2001.00222 (2020).

[118] E. Jul, H. Levy, N. Hutchinson, and A. Black. 1988. Fine-grained mobility in the Emerald system. *ACM Transactions on Computer Systems (TOCS)* 6, 1 (1988), 109–133.

[119] K. Kawaguchi, J. Huang, and L. P. Kaelbling. 2019. Effect of Depth and Width on Local Minima in Deep Learning. *Neural Computation* 31, 7 (2019), 1462–1498.

[120] A. Kejariwal, S. Kulkarni, and K. Ramasamy. 2017. Real Time Analytics: Algorithms and Systems. *The Computing Research Repository* abs/1708.02621 (2017).

[121] J. Khalid, E. Rozner, W. Felter, C. Xu, K. Rajamani, A. Ferreira, and A. Akella. 2018. Iron: Isolating Network-based CPU in Container Environments. In *iProceedings of the Symposium on Networked Systems Design and Implementation*. Renton, WA, 313–328.

[122] Y. Kim and J. Lin. [n.d.]. Serverless data analytics with Flint. In *IEEE International Conference on Cloud Computing*. 451–455.

[123] D. Kjerrumgaard. 2018. Real-Time Analytics with Pulsar Functions. (Nov. 2018). https://streaml.io/blog/eda-real-time-analytics-with-pulsar-functions.

[124] A. Klimovic, Y. Wang, C. Kozyrakis, P. Stuedi, J. Pfefferle, and A. Trivedi. 2018. Understanding Ephemeral Storage for Serverless Analytics. In *Proceedings of the USENIX Annual Technical Conference*. 789–794.

[125] A. Klimovic, Y. Wang, P. Stuedi, A. Trivedi, J. Pfefferle, and C. Kozyrakis. 2018. Pocket: Elastic ephemeral storage for serverless analytics. In *Proceedings of the USENIX Symposium on Operating Systems Design and Implementation*. 427–444.

[126] R. Koller and D. Williams. 2017. Will Serverless End the Dominance of Linux in the Cloud?. In *Proceedings of the 16th Workshop on Hot Topics in Operating Systems*. Whistler, BC, Canada, 169–173.

[127] J. Konečný, B. McMahan, D. Ramage, and P. Richtárik. 2016. Federated Optimization: Distributed Machine Learning for On-Device Intelligence. *The Computing Research Repository* abs/1610.02527 (2016).

[128] M. Król and I. Psaras. 2017. NFaaS: Named Function As a Service. In *Proceedings of the 4th ACM Conference on Information-Centric Networking*. Berlin, Germany, 134–144.

[129] D. Kumanov, L.-H. Hung, W. Lloyd, and K. Y. Yeung. 2018. Serverless computing provides on-demand high performance computing for biomedical research. *The Computing Research Repository* abs/1807.11659 (2018).

[130] D. Lazar, Y. Gilad, and N. Zeldovich. 2019. Yodel: Strong Metadata Security for Voice Calls. In *Proceedings of the 27th ACM Symposium on Operating Systems Principles*. Huntsville, Ontario, Canada, 211–224.

[131] B. D. Lee, M. A. Timony, and P. Ruiz. 2019. DNAvisualization.org: a serverless web tool for DNA sequence visualization. *Nucleic Acids Research* 47, W1 (6 2019), W20–W25.

[132] K. Lee, M. Lam, R. Pedarsani, D. Papailiopoulos, and K. Ramchandran. 2018. Speeding Up Distributed Machine Learning Using Codes. *IEEE Transactions on Information Theory* 64, 3 (March 2018), 1514–1529.

[133] Kai Li. 1988. IVY: A Shared Virtual Memory System for Parallel Computing. *ICPP (2)* 88 (1988), 94.

[134] L. Li, K. G. Jamieson, A. Rostamizadeh, E. Gonina, M. Hardt, B. Recht, and A. Talwalkar. 2018. Massively Parallel Hyperparameter Tuning. *The Computing Research Repository* abs/1810.05934 (2018).

[135] H. Lim, D. Han, D. G. Andersen, and M. Kaminsky. 2014. MICA: A Holistic Approach to Fast In-memory Key-value Storage. In *Proceedings of the Symposium on Networked Systems Design and Implementation*. Seattle, WA, 429–444.

[136] W. Lin, Z. Qian, J. Xu, S. Yang, J. Zhou, and L. Zhou. 2016. Streamscope: continuous reliable distributed processing of big data streams. In *Proceedings of the Symposium on Networked Systems Design and Implementation*. 439–453.

[137] Pedro García López, Marc Sánchez-Artigas, Gerard París, Daniel Barcelona Pons, Álvaro Ruiz Ollobarren, and David Arroyo Pinto. 2018. Comparison of FaaS Orchestration Systems. In *Proceedings of the IEEE/ACM International Conference on Utility and Cloud Computing Companion*. 148–153.

[138] S. Ma, L. Cui, D. Dai, F. Wei, and X. Sun. 2018. LiveBot: Generating Live Video Comments Based on Visual and Textual Contexts. *The Computing Research Repository* abs/1809.04938 (2018).

[139] A. Madhavapeddy, R. Mortier, C. Rotsos, D. Scott, B. Singh, T. Gazagnaire, S. Smith, S. Hand, and J. Crowcroft. 2013. Unikernels: Library Operating Systems for the Cloud. In *Proceedings of the 18th International Conference on Architectural Support for Programming Languages and Operating Systems*. 461–472.

[140] M. Malawski, A. Gajek, A. Zima, B. Balis, and K. Figiela. 2017. Serverless execution of scientific workflows: Experiments with HyperFlow, AWS Lambda and Google Cloud Functions. *Future Generation Computer Systems* (Nov. 2017).
[141] G. Malewicz, M. H Austern, A. J. C. Bik, J. C. Dehnert, I. Horn, N. Leiser, and G. Czajkowski. 2010. Pregel: a system for large-scale graph processing. In *Proceedings of the ACM SIGMOD International Conference on Management of data*. 135–146.
[142] G. Malewicz, M. H. Austern, A. J. C. Bik, J. C. Dehnert, I. Horn, N. Leiser, and G. Czajkowski. 2010. Pregel: A System for Large-scale Graph Processing. In *Proceedings of the 2010 ACM SIGMOD International Conference on Management of Data*. Indianapolis, IN, 135–146.
[143] F. Manco, C. Lupu, F. Schmidt, J. Mendes, S. Kuenzer, S. Sati, K. Yasukata, C. Raiciu, and F. Huici. 2017. My VM is Lighter (and Safer) Than Your Container. In *Proceedings of the 26th Symposium on Operating Systems Principles*. Shanghai, China, 218–233.
[144] A. McGregor. 2014. Graph Stream Algorithms: A Survey. *SIGMOD Record* 43, 1 (May 2014), 9–20.
[145] B. McMahan and D. Ramag. 2017. Federated Learning: Collaborative Machine Learning without Centralized Training Data. (2017). https://ai.googleblog.com/2017/04/federated-learning-collaborative.html.
[146] B. Mirzasoleiman, J. A. Bilmes, and J. Leskovec. 2019. Data Sketching for Faster Training of Machine Learning Models. *The Computing Research Repository* abs/1906.01827 (2019).
[147] P. Moritz, R. Nishihara, S. Wang, A. Tumanov, R. Liaw, E. Liang, M. Elibol, Z. Yang, W. Paul, M. I. Jordan, and I. Stoica. 2018. Ray: A Distributed Framework for Emerging AI Applications. In *Proceedings of the Symposium on Operating Systems Design and Implementation*. 561–577.
[148] S. Muthukrishnan. 2005. Data Streams: Algorithms and Applications. *Foundations and Trends in Theoretical Computer Science* 1, 2 (Aug. 2005).
[149] A. Nair, P. Srinivasan, S. Blackwell, C. Alcicek, R. Fearon, A. D. Maria, V. Panneershelvam, M. Suleyman, C. Beattie, S. Petersen, S. Legg, V. Mnih, K. Kavukcuoglu, and D. Silver. 2015. Massively Parallel Methods for Deep Reinforcement Learning. *The Computing Research Repository* abs/1507.04296 (2015).
[150] X. Niu, D. Kumanov, L.-H. Hung, W. Lloyd, and K. Y. Yeung. 2019. Leveraging Serverless Computing to Improve Performance for Sequence Comparison. In *Proceedings of the 10th ACM International Conference on Bioinformatics, Computational Biology and Health Informatics*. Niagara Falls, NY, 683–687.
[151] J. Ousterhout, P. Agrawal, D. Erickson, C. Kozyrakis, J. Leverich, D. Mazières, S. Mitra, . Narayanan, G. Parulkar, M. Rosenblum, et al. 2010. The Case for RAMClouds: Scalable High-performance Storage Entirely in DRAM. *ACM SIGOPS Operating Systems Review* 43, 4 (2010), 92–105.
[152] A. Passwater. 2018. 2018 Serverless Community Survey: huge growth in serverless usage. (Aug. 2018). https://serverless.com/blog/2018-serverless-community-survey-huge-growth-usage/.
[153] J. M. Phillips. 2016. Coresets and Sketches. *The Computing Research Repository* abs/1601.00617 (2016).
[154] W. Ping, K. Peng, A. Gibiansky, S. Ö. Arik, A. Kannan, S. Narang, J. Raiman, and J. Miller. 2017. Deep Voice 3: Scaling Text-to-Speech with Convolutional Sequence Learning. *The Computing Research Repository* abs/1710.07654 (2017).
[155] D. Pinto, J. P. Dias, and H. S. Ferreira. 2018. Dynamic Allocation of Serverless Functions in IoT Environments. *The Computing Research Repository* abs/1807.03755 (2018).
[156] Q. Pu, S. Venkataraman, and I. Stoica. 2019. Shuffling, fast and slow: scalable analytics on serverless infrastructure. In *Proceedings of the 16th USENIX Symposium on Networked Systems Design and Implementation*. 193–206.
[157] A. Rinberg, A. Spiegelman, E. Bortnikov, E. Hillel, I. Keidar, L. Rhodes, and H. Serviansky. 2019. Fast Concurrent Data Sketches. *The Computing Research Repository* abs/1902.10995 (2019).
[158] J. Sampé, G. Vernik, M. Sánchez-Artigas, and P. García-López. 2018. Serverless Data Analytics in the IBM Cloud. In *Proceedings of the 19th International Middleware Conference Industry*. Rennes, France, 1–8.
[159] Mark Seaborn and Thomas Dullien. [n.d.]. Exploiting the DRAM rowhammer bug to gain kernel privileges. "https://www.blackhat.com/docs/us-15/materials/us-15-Seaborn-Exploiting-The-DRAM-Rowhammer-Bug-To-Gain-Kernel-Privileges.pdf".
[160] Z. Shen, Z. Sun, G.-E. Sela, E. Bagdasaryan, C. Delimitrou, R. Van Renesse, and H. Weatherspoon. 2019. X-Containers: Breaking Down Barriers to Improve Performance and Isolation of Cloud-Native Containers. In *Proceedings of the Twenty-Fourth International Conference on Architectural Support for Programming Languages and Operating Systems*. Providence, RI, 121–135.
[161] S. Sievert. 2019. Better and faster hyperparameter optimization with Dask. (2019). https://blog.dask.org/2019/09/30/dask-hyperparam-opt.
[162] J. A. Silva, E. R. Faria, R. C. Barros, E. R. Hruschka, A. C. P. L. F. de Carvalho, and J. Gama. 2013. Data Stream Clustering: A Survey. *Comput. Surveys* 46, 1, Article 13 (July 2013), 13:1–13:31 pages.
[163] A. Singhvi, K. Houck, A. Balasubramanian, M. D. Shaikh, S. Venkataraman, and A. Akella. 2019. Archipelago: A Scalable Low-Latency Serverless Platform. *The Computing Research Repository* abs/1911.09849 (2019).
[164] A. Singhvi, J. Khalid, A. Akella, and S. Banerjee. 2019. SNF: Serverless Network Functions. *The Computing Research Repository* abs/1910.07700 (2019).
[165] S. Sivasubramanian. 2012. Amazon dynamoDB: A Seamlessly Scalable Non-relational Database Service. In *Proceedings of the ACM International Conference on Management of Data*. Scottsdale, AZ, 729–730.
[166] J. Snoek, H. Larochelle, R. P. AdamsM. Gabbrielli, S. Giallorenzo, I. Lanese, F. Montesi, M. Peressotti, and S. P. Zingaro. [n.d.]. ([n. d.]).
[167] S. Soltesz, H. Pötzl, M. E. Fiuczynski, A. Bavier, and L. Peterson. 2007. Container-based Operating System Virtualization: A Scalable, High-performance Alternative to Hypervisors. In *Proceedings of the 2Nd ACM SIGOPS/EuroSys European Conference on Computer Systems 2007*. Lisbon, Portugal, 275–287.
[168] V. Sreekanti, C. Wu, X. C. Lin, J. S.-Smith, J. M. Faleiro, J. E. Gonzalez, J. M. Hellerstein, and A. Tumanov. 2020. Cloudburst: Stateful Functions-as-a-Service. *The Computing Research Repository* abs/2001.04592 (2020).
[169] Emil Stefanov, Marten Van Dijk, Elaine Shi, Christopher Fletcher, Ling Ren, Xiangyao Yu, and Srinivas Devadas. 2013. Path ORAM: an extremely simple oblivious RAM protocol. In *CCS*.
[170] V. Strassen. 1969. Gaussian elimination is not optimal. *Numer. Math.* 13, 4 (Aug 1969), 354–356.
[171] S. Sun, W. Chen, L. Wang, X. Liu, and T.-Y. Liu. 2015. On the Depth of Deep Neural Networks: A Theoretical View. *The Computing Research Repository* abs/1506.05232 (2015).
[172] N. Sundaram, N. Satish, M. M. A. Patwary, S. R. Dulloor, M. J. Anderson, S. G. Vadlamudi, D. Das, and P. Dubey. 2015. GraphMat: High Performance Graph Analytics Made Productive. *Proceedings of the VLDB Endowment* 8, 11 (July 2015), 1214–1225.
[173] L. Toader, A. Uta, A. Musaafir, and A. Iosup. 2019. Graphless: Toward Serverless Graph Processing. In *Proceedings of the 18th International Symposium on Parallel and Distributed Computing*. 66–73.
[174] A. Uta, S. Au, A. Ilyushkin, and A. Iosup. 2018. Elasticity in Graph Analytics? A Benchmarking Framework for Elastic Graph Processing. In *Proceedings of the IEEE International Conference on Cluster Computing*. 381–391.
[175] J. van den Hooff, D. Lazar, M. Zaharia, and N. Zeldovich. 2015. Vuvuzela: Scalable Private Messaging Resistant to Traffic Analysis. In *Proceedings of the 25th Symposium on Operating Systems Principles*. Monterey, CA, 137–152.
[176] E. van Eyk, L. Toader, S. Talluri, L. Versluis, A. Uta, and A. Iosup. 2018. Serverless is More: From PaaS to Present Cloud Computing. *IEEE Internet Computing* 22, 05 (2018), 8–17.
[177] Abhishek Verma, Luis Pedrosa, Madhukar R. Korupolu, David Oppenheimer, Eric Tune, and John Wilkes. 2015. Large-scale cluster management at Google with Borg. In *Proceedings of the European Conference on Computer Systems (EuroSys)*. Bordeaux, France.
[178] T. Wagner. 2019. Serverless Networking is the next step in the evolution of serverless. (Oct. 2019). https://read.acloud.guru/https-medium-com-timawagner-serverless-networking-the-next-step-in-serverless-evolution-95bc8adaa904.
[179] A. Wang, H. Zhou, W. Xu, and X. Chen. 2017. Deep Neural Network Capacity. *The Computing Research Repository* abs/1708.05029 (2017).
[180] L. Wang, M. Li, Y. Zhang, T. Ristenpart, and M. Swift. 2018. Peeking Behind the Curtains of Serverless Platforms. In *Proceedings of the USENIX Annual Technical Conference*. Boston, MA, 133–146.
[181] S. Werner, J. Kuhlenkamp, M. Klems, J. Müller, and S. Tai. 2018. Serverless Big Data Processing using Matrix Multiplication as Example. In *Proceedings of the IEEE International Conference on Big Data*. 358–365.
[182] R. Wolski, C. Krintz, F. Bakir, G. George, and W.-T. Lin. 2019. CSPOT: Portable, Multi-scale Functions-as-a-service for IoT. In *Proceedings of the 4th ACM/IEEE Symposium on Edge Computing*. Arlington, VA, 236–249.
[183] Z. Xiong, X.-Y. Liu, S. Zhong, H. Yang, and A. Elwalid. 2018. Practical Deep Reinforcement Learning Approach for Stock Trading. *The Computing Research Repository* abs/1811.07522 (2018).
[184] M. Yan, P. Castro, P. Cheng, and V. Ishakian. 2016. Building a Chatbot with Serverless Computing. In *Proceedings of the 1st International Workshop on Mashups of Things and APIs*. Trento, Italy, 5:1–5:4.
[185] M. Zaharia, T. Das, H. Li, T. Hunter, S. Shenker, and I. Stoica. 2013. Discretized Streams: Fault-tolerant Streaming Computation at Scale. In *Proceedings of the ACM Symposium on Operating Systems Principles*. Farminton, PA, 423–438.
[186] M. Zhang, C. Krintz, M. Mock, and R. Wolski. 2019. Seneca: Fast and Low Cost Hyperparameter Search for Machine Learning Models. In *Proceedings of the IEEE 12th International Conference on Cloud Computing*. 404–408.
[187] M. Zhang, Y. Zhu, C. Zhang, and J. Liu. 2019. Video Processing with Serverless Computing: A Measurement Study. In *Proceedings of the 29th ACM Workshop on Network and Operating Systems Support for Digital Audio and Video*. Amherst, MA, 61–66.
[188] J. Zhao, G. Qiu, Z. Guan, W. Zhao, and X. He. 2018. Deep Reinforcement Learning for Sponsored Search Real-time Bidding. In *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*. London, UK, 1021–1030.
[189] C. Zhong, M. C. Gursoy, and S. Velipasalar. 2019. Deep Actor-Critic Reinforcement Learning for Anomaly Detection. *The Computing Research Repository* abs/1908.10755 (2019).

BRIEF BIOGRAPHIES

Anurag Khandelwal is an assistant professor at Yale University. He earned a Ph.D. from UC Berkeley. His research interests span distributed systems, networking and algorithms. In particular, his research focuses on addressing core challenges in distributed systems through novel algorithm and data structure design. During his Ph.D. at UC Berkeley, Anurag built large-scale data-intensive systems such as Succinct and Confluo, that led to deployments in several production clusters.

Arun Kejariwal is an engineering leader at Facebook Inc. where his team incessantly pushes the performance, scalability and efficiency envelope of the entire product portfolio of the company. The business impact driven thus far is over $100M. Previously, he was he was a R&D leader at MZ Inc.Arun earned a Ph.D. in computer science from University of California, Irvine.

Karthik Ramasamy is a Senior Director at Splunk Inc. Previously, he was the cofounder of Streamlio (acquired by Splunk), a company building next-generation real-time processing engines. Previously, he was engineering manager and technical lead for real-time analytics at Twitter, where he was the co-creator of Heron. Karthik holds a Ph.D. in computer science from the University of Wisconsin-Madison with a focus on databases.