A Survey on Cloud-Native and Serverless for ML/DL Workloads

Imran Matin
University of California San Diego
La Jolla, California, USA
imatin@ucsd.edu

ABSTRACT

As machine learning and deep learning (ML/DL) continues to flourish in both academia and industry, the demand for supporting ML/DL workloads effectively has increased dramatically. In this paper we perform a survey on the landscape of cloud-native and serverless frameworks for ML/DL workloads. We first provide an overview of distributed ML/DL and the challenges that ML/DL users face today in the ML/DL workflow. Next, for both cloud-native and serverless, we perform the following: describe its execution model, describe a state-of-the-art offering, compare the offerings in that field, and discuss research challenges and future directions. Serverless computing for ML/DL workloads is feasible for certain types of ML/DL workloads today and future improvements in the field will increase its ability to generalize to a wider range of workloads. Cloud-native services for ML/DL workloads allow for enterprises to quickly enter the ML/DL space with little to no ML/DL knowledge.

CCS CONCEPTS

• **Computing methodologies** → **Artificial intelligence**; **Machine learning**; **Distributed computing methodologies**; • **General and reference** → **Surveys and overviews**.

KEYWORDS

cloud computing, serverless, machine learning, deep learning, survey, distributed computing

ACM Reference Format:
Imran Matin. 2021. A Survey on Cloud-Native and Serverless for ML/DL Workloads. In *CSE234 '21: Data Systems for Machine Learning, December 02, 2021, La Jolla, CA.* ACM, New York, NY, USA, 10 pages. https://doi.org/10.1145/1122445.1122456

1 INTRODUCTION

1.1 What is ML/DL?

Machine learning and deep learning (ML/DL) are methods for using data and algorithms to learn to make better predictions. Today, we have seen the adoption of ML/DL in a wide variety of fields such as computer vision, speech recognition, and medicine [18]. The goal of ML/DL is to find a model that, given some feature vector x as input, is able to correctly predict the label y. The difference between ML and DL stems from the fact that DL can perform better on data where the inputted feature vector x is not intuitive. This is demonstrated by the difference in difficulty when extracting features for prediction from a labeled relational dataset such as a table compared to a labeled unstructured dataset such as a set of images.

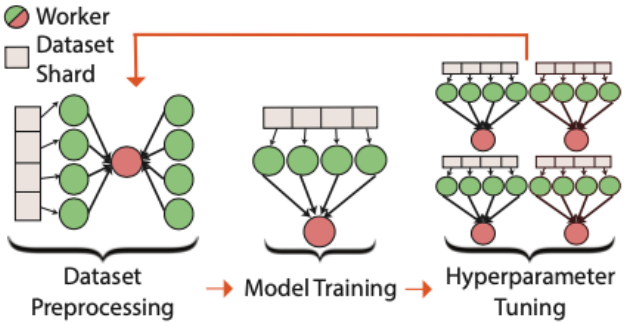

Figure 1: Typical end-to-end ML/DL Workload. Diagram by Carreira, et al.

1.2 What are ML/DL workloads?

ML/DL workloads are composed of three stages: dataset preprocessing, training, and hyperparameter search [19]. These stages are also commonly referred to as the ML/DL workflow. Each of these stages has different computational requirements which increases the complexity of resource provisioning for these types of workloads. In the dataset preprocessing stage, the goal is to use dataset transformations to prepare the dataset in such a way that is best suited for model training. Some commonly used dataset transformations are min-max scaling, standardization, and feature hashing, and are usually executed by large MapReduce jobs on the dataset. In the training stage, the goal is to use a loss function and a training dataset to find a model that minimizes the loss across the training set. The most commonly used optimization algorithm to do this is Stochastic Gradient Descent (SGD). In the hyperparameter search stage, the goal is to identify model parameters that achieve the best accuracy. The most commonly used method to do this is performing a grid search over the multi-dimensional parameter space.

1.3 What is an ML/DL framework?

A ML/DL framework is considered to be any tool, interface, or library that assists in the development of ML models without having to re-implement low-level algorithms. Today there exists many ML/DL frameworks and the most common ones are TensorFlow

CSE234 '21, December 02, 2021, La Jolla, CA

ACM ISBN 978-1-4503-XXXX-X/18/06…$15.00
https://doi.org/10.1145/1122445.1122456

and PyTorch. For an ML/DL framework to be considered end-to-end, it must be able to support all three stages of the ML/DL workflow.

1.4 What is distributed ML/DL?

Today, ML/DL workloads have become large enough to where computation cannot be done efficiently on a single node. Hence, computation is required to be scaled out to multiple nodes in a distributed fashion. This distribution of computation introduces interesting computational challenges in efficiency, synchronization, and more. The MapReduce distributed programming model has been widely adopted for processing large datasets and has been implemented by tools such as Hadoop [5] and Apache Spark [6]. Distributed SGD has been widely adopted for scalable model training as it allows workers to independently and concurrently execute SGD on a mini-batch of data and then asynchronously update the global model. Distributed model training is most commonly implemented using the parameter server architecture with clusters of bare-metal (i.e. physical hardware) or virtualized (i.e. virtual machines) compute resources alongside TensorFlow [13].

Distributed ML/DL can be categorized into four domains: its distributed optimization algorithms, its communication channels, the communication protocols/primitives, and its synchronization protocols. Distributed optimization algorithms such as distributed SGD are used for model training. Communication channels are shared storage used for intermediate storage of results. Communication protocols/primitives are used to implement the synchronization protocols for performing model updates during training [18].

1.5 What is cloud computing?

Cloud computing was popularized in the mid-2000s and its goal was to allow users to access shared computing and storage resources on-demand. Today the major cloud computing providers are Amazon Web Services (AWS) [2], Google Cloud Platform (GCP) [17], and Microsoft Azure [22]. The technology that drives cloud computing is virtualization as it enables a physical resource to be abstracted into multiple virtual resources that share the underlying physical hardware. These virtualized computing resources, which include networking, storage, security, etc., are supported by rich APIs and have become known as Infrastructure-as-a-Service (IaaS). The benefits of cloud computing are numerous however due to its "pay-as-you-go" billing model, computing expenses can grow quickly and unexpectedly. The ML/DL community has benefited greatly from the advent of cloud computing as well as increased access to data. ML/DL users can now execute the different, computationally demanding stages of the ML/DL workflow on virtual machines (VM) supported by scalable data stores all in the cloud.

1.6 What challenges and problems do ML users face with the ML/DL workflow today?

Current ML/DL frameworks were designed with an assumption that ML/DL users only leverage long-lived, coarsely managed clusters of VMs as infrastructure to execute the stages of the ML/DL workflow. This assumption along with the dramatically increasing scale and complexity of the ML/DL workflow has created challenges for ML/DL users that greatly hinder their productivity and effectiveness. The first challenge is that ML/DL users are required to manually configure numerous system-level parameters, such as number of workers/parameter servers, memory allocation, number of CPUs, physical topology, etc. The second challenge is that ML/DL users must also manage numerous ML-specific parameters, such as learning rate, learning algorithms, neural network structure and these parameters interact in non-obvious ways with the system-level parameters. The third challenge is that the different stages of the ML/DL workflow have differing computational requirements between and within themselves, and ML/DL users must account for this. The byproduct of these challenges are two problems: overprovisioning and resource management.

Table 1: Resource management responsibilities for ML/DL users. Table by Carreira, et al.

User responsibility	Description
Sharding data	Distribute datasets across VMs
Configuring storage systems	Setup a storage system (e.g. NFS)
Configuring OS/drivers	Choosing OS and drivers
Deploying frameworks	Install ML training frameworks
Monitoring	Monitor VMs for errors
Choosing VM configuration	Choosing VM types
Setup network connections	Make VMs inter-connect
Upgrading systems	Keep libraries up-to-date
Scaling up and down	Adapt to workload changes

ML/DL users are tempted to overprovision resources based on peak consumption requirements due to the complexity of their workloads which leads to the underutilization of resources and increased cost during the ML/DL workflow. The second problem of resource management stems from the fact that low-level VM resources are exposed to ML/DL users even though these users are usually only trained in the ML/DL aspects of the ML/DL workflow. The management of resources for the ML/DL workflow is not trivial and Table 1 highlights some of the responsibilities ML/DL users have today [9].

2 SERVERLESS FOR ML/DL WORKLOADS

2.0.1 What is serverless computing? Serverless computing is a novel, cloud computing execution model that has come into the foreground during the past decade. Today, there are many commercial serverless offerings such as AWS Lambda [3], Google Cloud Functions [16], and Microsoft Azure Functions [7] which provide the abstract serverless computing model with a "pay-for-what-you-use" billing model. Serverless computing allows developers to focus on the business logic of their applications, while the cloud provider takes on the burden of provisioning and managing computing resources. To use the serverless model, a developer simply needs to specify the function they want to execute and the cloud provider will provision the required ephemeral resources on-demand to execute it. Serverless computing is also commonly called Functions-as-a-Service (FaaS) as the smallest computational unit in the serverless computing model is a stateless lambda function. The FaaS billing model differs from the IaaS billing model as costs are only incurred based on what resources are actually used rather than what resources have been reserved.

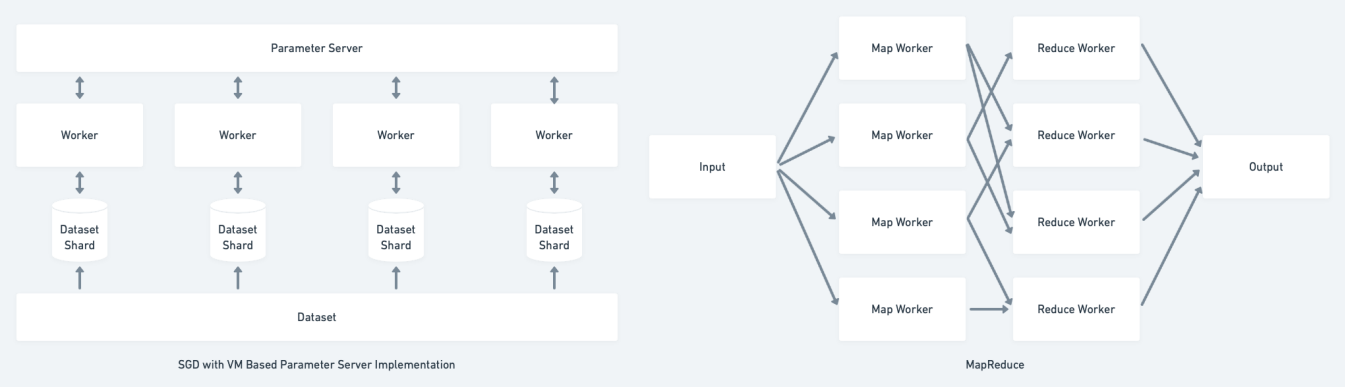

Figure 2: SGD and MapReduce.

2.0.2 What are the benefits of serverless computing? The benefits of serverless computing are numerous. The billing model is based on actual resource consumption which helps avoid the problem of incurring costs for reserved but unused resources. The interface is very simple as all developers need to do is write their code and submit it for execution. It is naturally elastic with lower startup and setup overheads and is able to scale up to thousands of concurrently executing functions. It has built in mechanisms to avoid over-provisioning as it is based on restricted, stateless lambda functions with fine spatial (i.e. small resource requirements) and temporal granularity (i.e. short execution time). It lifts the burden of manually configuring, deploying, and managing long term computing resources such as VMs off of developers and onto the cloud provider.

2.0.3 Why serverless computing for ML/DL workloads? Originally, serverless computing was meant for web microservices and IoT, but has recently been moving towards being used in data intensive applications. In the case of ML/DL workloads, the serverless computing model inherently addresses the problems of over-provisioning and resource management due to FaaS' ability to scale, its fine-grained stateless lambda functions, and its cloud managed infrastructure approach. Also, since ML/DL workloads have a high degree of parallelization, the concurrent serverless execution model fits right in.

2.0.4 What are the constraints of serverless for ML/DL workloads? Unfortunately, most of the assumptions and design decisions made by existing ML/DL frameworks are orthogonal to the design principles of FaaS. Serverless functions have small local resource constraints in terms of memory, storage, and network bandwidth, but ML/DL frameworks assume that abundant resources are available for use. P2P communication between serverless functions is not supported today, but ML/DL frameworks require it to support communication frameworks like the Message Passing Interface (MPI). Serverless functions have short execution times and unpredictable launch times, but ML/DL frameworks assume compute units are long-lived. Cloud providers do not support customized scaling and scheduling strategies for serverless functions yet. Serverless functions require an intermediate data store for worker communication and model updates since they are stateless and cloud providers do not provide a low latency, high throughput data store specialized for ML/DL workloads that serves this purpose yet.

2.0.5 How are serverless frameworks for ML/DL workloads generally architected? All serverless frameworks for ML/DL workloads share a similar architecture and set of core components. Figure 3 visually depicts this architecture and its core components. A Python API must be provided to ML/DL users for using the framework. A manager service must be implemented to service API requests, schedule, track, and manage the worker nodes performing the computation for jobs, and store results to external storage. An intermediate datastore must be used to support worker communication and global model updates. A lightweight worker runtime must be provided to encapsulate source code used when executing jobs on serverless functions. An external storage source must be provided to store results and datasets. These are the core components of the architecture for a serverless framework for ML/DL workloads, but other optimizations are required to ensure the framework is efficient enough to be usable.

2.1 State-of-the-Art (SOTA)

2.1.1 What is the SOTA for serverless ML/DL workloads? Serverless for ML/DL workloads is a relatively new research field, and there does not exist a state-of-the-art serverless framework for ML/DL workloads today.

2.1.2 What features does a SOTA serverless ML/DL framework have? A state-of-the-art serverless ML/DL framework for ML/DL workloads would have many features that enhance ML/DL users' abilities in the ML/DL workflow. It must have a rich UI and Python API that is easy to use, allows for seamless integrations, and supports visualizations. It must support all stages of the ML/DL workflow as well as be highly scalable and customizable. Scheduling should be configurable based on different goals including total training time, total cost or both. The configurations of resources such as memory, storage, and compute (CPU, GPU, TPU, etc.) should be customizable if desired by the user. Multiple clouds should be supported including AWS , GCP, and Azure. A fully serverless or hybrid model with a parameter server for execution should be supported. It should support adaptive fine-grained resource allocation during

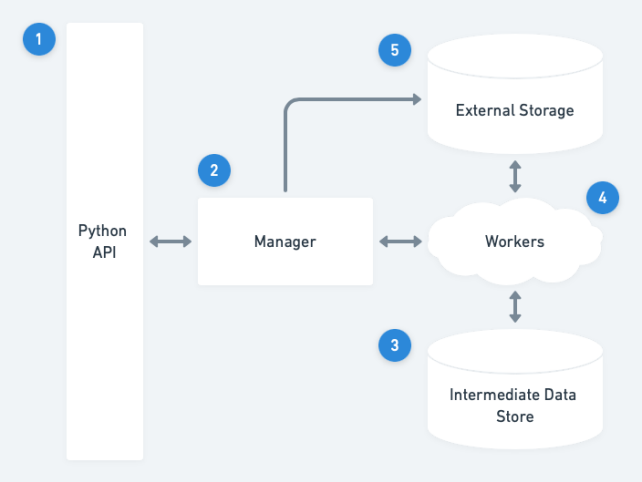

Figure 3: General architecture of a serverless framework for ML/DL workloads.

job execution. It should support a low-latency, high throughput distributed data store for ML/DL communication, intermediate results, and model updates. It should provide an ultra-light customizable runtime that supports many ML/DL frameworks (e.g. TensorFlow, PyTorch, MXNet, etc.) and distributed optimization algorithms (e.g. distributed SGD, distributed ADMM, etc.). It should support multiple communication channels (e.g. Redis, general-purpose persistent storage). It should support multiple communication patterns/primitives (e.g. ScatterReduce, AllReduce, etc.) for worker communication and model updates. It should support multiple synchronization protocols, asynchronous and synchronous, such as Bulk Synchronous Parallel (BSP), Stale Synchronous Parallel (SSP), and Hybrid Synchronous Parallel (HSP). It should support many model families such as General Linear Models (GLMs), Tree Learners, and Artificial Neural Networks (ANNs).

2.2 Deep Dive

2.2.1 LambdaML. The LambdaML paper [18] provides a systematic, comparative study of distributed ML/DL training over FaaS and IaaS. The work also implements a prototype named LambdaML which is a FaaS-based ML/DL system built on top of AWS Lambda. The paper defines four major dimensions of distributed ML/DL systems: the optimization algorithm, the communication channel, the communication pattern, and the synchronization protocol. The LambdaML system is designed with these dimensions in mind and is then evaluated against well optimized IaaS systems such as distributed PyTorch, Angel, and a hybrid parameter server implementation to understand the benefits of FaaS for ML/DL.

LambdaML is implemented on AWS, leverages the AWS UI console for its UI, and provides a Python API. It is not an end-to-end system as it is only designed for one stage of the ML workflow, model training. Its scalability, performance, and resource customizability is limited by what is provided by AWS' services. It supports both a fully serverless and a hybrid parameter server model for execution. Custom resource scheduling based on metrics such as cost and training time is not supported. Adaptive fine-grained resource allocation is not supported. It has a lightweight runtime that supports PyTorch, GLMs such as linear regression, sparse linear regression, SVM, sparse SVM, KMeans, sparse KMeans, and ANNs such as MobileNet and ResNet. It supports the distributed optimization algorithms of gradient averaging SGD, model averaging SGD, and ADMM. It supports AWS S3, ElastiCache, or DynamoDB as communication channels. It supports AllReduce, ScatterReduce, and asynchronous communication patterns/primitives. It supports a custom synchronous protocol as well as an asynchronous protocol.

The paper's main contributions are the insights it provides about the tradeoffs between FaaS and IaaS systems for ML/DL workloads. The first insight is that FaaS can be faster than IaaS but only if the underlying workload is made communication efficient. The second insight is that even when FaaS is much faster, it is not much cheaper. These insights are also demonstrated in the paper's analytical model that captures the cost/performance tradeoff for FaaS.

2.2.2 Cirrus. The Cirrus paper [9] provides an implementation of an end-to-end serverless framework for ML/DL workloads. The Cirrus system is evaluated against PyWren, TensorFlow, and Bosen.

Cirrus is implemented on AWS, provides a custom UI dashboard for visualizations, and provides a Python API. It is an end-to-end system as it supports all stages of the ML/DL workflow including dataset preprocessing, feature engineering, model training, and parameter tuning. Its scalability, performance, and resource customizability is limited by what is provided by AWS' services. It uses a hybrid parameter server model for execution. Custom resource scheduling based on metrics such as cost and training time is not supported for model training, but for hyperparameter search, early termination of configurations using the UI dashboard is supported. Adaptive fine-grained resource allocation is not supported. It provides a lightweight runtime with smart iterators that will prefetch and buffer mini-batches into local memory in parallel with currently executing computations as well as supports sharding, multithreading, compression, and filters in its distributed data store. It has a lightweight runtime implemented in C++ and provides general abstractions for ML computations as well as data abstractions for implementing new models. Out-of-the-box it supports GLMs such as sparse linear regression, softmax, and LDA as well as collaborative filtering. It supports the distributed optimization algorithms of SGD, Adagrad, and Momentum. It provides a custom distributed data store powered by a parameter server as its communication channel. It supports asynchronous communication patterns/primitives and an asynchronous synchronization protocol.

The paper provides two main contributions. The first is that it demonstrates serverless frameworks for ML/DL workloads must be specialized and carefully designed for ML/DL to achieve acceptable levels of efficiency. The second is that it demonstrates that serverless frameworks for ML/DL workloads are feasible and can greatly reduce the complexity of the ML/DL workflow for ML/DL users.

2.2.3 Siren. The Siren paper [28] provides a new synchronization protocol named Hybrid Synchronous Parallel (HSP) as well as implements a deep reinforcement learning (DRL) dynamic resource scheduler for serverless ML/DL workloads. The Siren system is evaluated against MXNet implemented on AWS EC2 clusters.

Table 2: Comparison across standard dimensions of serverless frameworks for ML/DL workloads.

	SOTA	LambdaML	Cirrus	Siren	PyWren
Multi-Cloud supported?	Yes	No, only AWS	No, only AWS	No, only AWS	Yes
UI provided?	Yes	Yes, AWS Web UI	Yes, custom dashboard	No	No
Python API provided?	Yes	Yes	Yes	Yes	Yes
Designed for end-to-end ML/DL workflow?	Yes	No, only model training	Yes	No, only model training	Yes
How scalable is it?	Infinitely	Only what AWS' services provide	Only what AWS' services provide	Only what AWS' services provide	Only what Cloud Provider provides
Resources customizable?	Yes	Only what AWS' services provide	Only what AWS' services provide	Only what AWS' services provide	Only what Cloud Provider provides
Fully serverless or hybrid parameter server execution model supported?	Both	Both	Hybrid parameter server	Fully serverless	Fully serverless
Customizable resource scheduling?	Yes, by cost, training time, or both	No	Yes, only for hyperparameter search	Yes, by cost and training time	No
Adaptive fine-grained resource allocation supported?	Yes	No	No	Yes, uses DRL dynamic resource scheduler	No
Ultra-lightweight, customizable runtime provided?	Yes	Yes	Yes	Yes	Yes
Supported ML/DL frameworks	TensorFlow, PyTorch, MxNet, etc.	PyTorch	Custom C++ framework	MXNet	N/A
Supported ML/DL Model families	General Linear Models (GLMs), Tree Learners, Artificial Neural Networks (ANNs), etc.	GLMs, ANNs	GLMs, collaborative filtering	GLMs, ANNs	N/A
Supported optimization algorithms	SGD, ADMM, etc.	Gradient averaging SGD, model averaging SGD, ADMM	SGD, Adagrad, Momentum	SGD	N/A
Supported communication channels	External storage, in-memory key-value store, parameter server, etc.	AWS S3, ElastiCache, DynamoDB	Custom distributed, parameter-server based data store	AWS S3	AWS S3, GCP GCS, Azure Storage, Redis
Supported communication patterns/primitives	AllReduce, ScatterReduce, etc.	Reduce, ReduceScatter, asynchronous	Asynchronous	Asynchronous	MapReduce
Supported synchronization protocols	BSP, SSP, HSP, asynchronous, etc.	Custom synchronous protocol, asynchronous	Asynchronous	HSP	BSP, asynchronous

Siren is implemented on AWS, does not provide a UI, and provides a Python API. It is not an end-to-end system as it is only designed for one stage of the ML workflow, model training. Its scalability, performance, and resource customizability is limited by what is provided by AWS' services. It supports a fully serverless model for execution. Custom resource scheduling and adaptive fine grain resource allocation based on training time and cost are supported by the system's DRL dynamic resource scheduler. It has a lightweight runtime that supports MXNet APIs for model building and training including GLMs and ANNs. It supports the distributed optimization algorithm SGD. It supports AWS S3 as its communication channel. It supports an asynchronous communication pattern/primitive. It uses HSP as its synchronization protocol.

The paper's main contributions are its HSP synchronization protocol and its DRL dynamic resource scheduler. HSP is a computing mode where within each epoch, all Lambda functions will update the model asynchronously; however at the end of each epoch there is a synchronization barrier imposed. HSP enables the paper's DRL dynamic resource scheduler to dynamically control the number and memory size of the stateless functions that should be used in each training epoch with an objective of minimizing possible training time given some cost.

2.2.4 PyWren. The PyWren paper [20] provides a data processing system that is architected using stateless functions with remote storage and relies on the MapReduce distributed programming model's map primitive for computation. It is evaluated against monolithic MapReduce jobs running on AWS EC2 compute resources.

PyWren was originally implemented solely on AWS Lambda, but has been expanded to support AWS, GCP, and Azure. It is an end-to-end system as it supports all stages of the ML/DL workflow since all of these stages can be modeled as MapReduce jobs. Its scalability, performance, and resource customizability is limited by what is provided by the cloud provider's services. It supports a fully serverless execution model. Custom resource scheduling based on metrics such as cost and training time is not supported. Adaptive fine-grained resource allocation is not supported. It has a lightweight runtime that supports Python only. It does not claim to support any specific ML/DL frameworks or models but technically it is able to support frameworks that are implemented in Python. It supports AWS S3 as its communication channel. It supports MapReduce as its communication pattern/primitive. It supports the synchronous protocol BSP as well as an asynchronous protocol.

The paper's main contributions are its reimplementation of the MapReduce distributed programming model with a serverless architecture, and its discussion of the different systems aspects that affect the serverless execution model's viability. It is a different system compared to the other systems we have reviewed in this paper as it implements the map primitive on top of AWS Lambda and uses AWS S3 as external storage for input and output. This map primitive can then be used to implement more complex abstractions to support the different stages of the ML/DL workflow. It also discusses the challenges that serverless faces in terms of resource balancing, pricing, scalable scheduling, debugging, distributed storage, launch overheads, and support for specialized hardware.

2.2.5 Exploring Serverless Computing for Neural Network Training. Fang et. Al. [21] have explored potential optimizations for training neural networks using serverless computing. In this work, a hybrid serverless model is used to study the effects of using a multi-layer parameter server structure to reduce the latency of synchronous gradient transfers from workers to the parameter server during training. The authors explore the idea of a multi-layer parameter server structure with an offline mathematical model. They have also proposed two online gradient descent methods for 1) finding the optimal memory size of each severless instance such that its monetary cost is optimized and 2) maximizing the performance-cost ratio of each serverless instance.

The results of their experiments demonstrate that it is necessary to minimize the frequency and quantity of data transfer between subsequent serverless instances to improve the efficiency training of neural networks. This work only applies to synchronous model updates and does not offer an end-to-end framework for ML/DL workloads as it is focused on exploring optimizations for model training using serverless computing.

2.2.6 OverSketched Newton: Fast Convex Optimization for Serverless Systems. Gupta et Al. [27] have explored developing a new randomized, distributed, Hessian-based optimization algorithm to solve large-scale convex optimization problems in serverless systems named OverSketched Newton. The OverSketched Newton optimization algorithm contributes solutions for straggler-resilient Hessian calculations and straggler mitigation during gradient calculation for serverless computing. The optimization algorithm is implemented on AWS Lambda using PyWren.

The main contribution of this work is the OverSketched Newton algorithm which is specifically designed for serverless distributed optimization. It provides a deep theoretical analysis of the algorithm as well as its experimental results.

2.3 Research Challenges

2.3.1 What are the common challenges faced by serverless frameworks for ML/DL workloads today? Serverless computing for ML/DL workloads is feasible and preferable in many ways, but as a new research direction there still exists many challenges.

Cloud providers do not support specialized hardware such as GPUs which hinders the practicality of using serverless for DL workloads. GPUs have not been multiplexed in the way that CPUs have been before, and networks will need to be able to efficiently support the transfer of models with large amounts of parameters during computation.

Cloud providers do not provide users with information about the infrastructure serverless functions are executed on, hence customized scheduling solutions are difficult to implement. Hence, serverless frameworks must implement scheduling strategies with novel techniques such as the DRL dynamic resource scheduler of Siren.

Cloud providers do not provide a low latency, high throughput storage solution that is specialized for ML/DL workloads. Hence, serverless frameworks must rely on solutions such as general-purpose persistent storage and in-memory key value stores such as the distributed data store used in Cirrus.

From the results in the LambdaML paper, it is clear that serverless frameworks are not the most performant solution today for ML/DL workloads with high communication requirements. The

serialization/deserialization operation as well as the network bandwidth of serverless functions create a bottleneck for jobs with heavy communication.

2.3.2 Where is the field going? The field of serverless computing for ML/DL workloads is relatively new as the earliest paper reviewed in this survey was released in 2017. From the research challenges above, it is clear that cloud providers will play a large role in expanding the capabilities of serverless frameworks for ML/DL workloads. New innovations in hardware, networks, and software are required to continue increasing the performance of serverless computing. New studies on when to use FaaS vs IaaS solutions depending on the types of workload will be required. Most importantly of all, the Systems and AI/ML communities will need to continue collaborating to solve the ever growing complexity, scale, and demand of ML/DL workloads across all domains.

3 CLOUD-NATIVE FOR ML/DL WORKLOADS

3.0.1 What is Cloud-Native? The cloud computing paradigm has enabled a wide variety of technological advances to occur in society. Microsoft Azure defines cloud-native architectures and technologies as an approach to designing, constructing, and operating workloads that are built in the cloud and take full advantage of the cloud computing model [24]. Hence, cloud-native represents the idea of moving workloads away from on-premise hardware resources to the cloud. Today, the cloud computing space is dominated by three platforms: Amazon Web Services (AWS), Google Cloud Platform (GCP), and Microsoft Azure. These platforms offer a wide array of solutions for customers and are trusted in the industry. Other cloud computing platforms exist including IBM Watson Studio, H2O, and DataRobot, but they are not as fully featured as the three main platforms.

3.0.2 What are the benefits of Cloud-Native? Cloud-native technologies are centered around the managed services model. Managed services push the difficult work of setting up and maintaining a service onto the cloud provider and away from the user. Due to the shift of responsibilities onto the cloud provider, the user is able to focus on using the service for its business objectives rather than on maintaining the service itself. The managed service model is common in the computing industry and some examples of it include: Platform-as-a-Service (PaaS), Functions-as-a-Service (FaaS), Infrastructure-as-a-Service (IaaS), and even ML-as-a-Service (MLaaS).

3.0.3 Why Cloud-Native for ML/DL workloads? MLaaS has greatly facilitated the increase in adoption of ML/DL solutions for many enterprises. It has significantly lowered the expertise required to create ML/DL solutions while simultaneously simplifying the ML/DL workflow. Cloud providers offer managed services for every step of the ML/DL workflow at an even finer granularity than the three stage workflow discussed above. Cloud providers have segmented the three stage workflow into an eight stage workflow that includes: data collection, data processing, feature engineering, data labelling, model design, training, optimization, and deployment and monitoring [10]. The main benefit of these cloud-native managed services is that customers can now use higher level abstractions provided by the cloud provider to execute the ML/DL workflow with minimal ML/DL expertise. For small enterprises and large enterprises with little ML/DL experience, MLaaS enables rapid exploration at a lower cost when compared to the overhead of hiring a complete ML/DL team [8].

3.0.4 What are the constraints of Cloud-Native for ML/DL workloads? Cloud-Native technologies for ML/DL workloads are still relatively new and they are iterated on constantly. One constraint of these types of technologies for ML/DL workloads is that they can become very sticky. Cloud platforms want to keep their customers on their cloud and this is done using a technique called vendor lock-in. Once customers are using many cloud-native services from a single cloud provider, it can be difficult to quickly and efficiently migrate away from the specific cloud they are using. Another constraint of cloud native for ML/DL workloads is that it can be difficult to customize the managed service to one's needs. Cloud-Native managed services are limited by what the cloud provider supports, hence if one wants to "bring-their-own" custom solutions to the cloud, then they may not be able to leverage all of the features of Cloud-Native for ML/DL workloads. For example, customers who want to use a cloud-native service for one stage of their ML/DL workload and a custom solution for another stage in the workload may find it difficult to do so due to tight coupling between the cloud provider's cloud-native services. Finally, by using cloud-native services for ML/DL workloads, customers are forced to pay potentially higher costs compared to if they were to implement their own solutions independently.

3.0.5 How are Cloud-Native for ML/DL workloads generally architected? The main cloud computing providers all provide similar cloud-native services for ML/DL workloads. Generally, there exists a managed service for each stage of the ML/DL workflow. If possible, the cloud provider will support customization for each stage of the ML/DL workflow such as during model training or dataset preprocessing. Customers begin by uploading their datasets to cloud storage. Once the dataset is available, they will use the cloud-provider's managed services to pre-process and label their datasets. Next, customers will use the cloud-provider's AutoML solutions or UI to design, build and train their models. It is important to note that building and training a custom model is usually supported, but can sometimes be difficult to do within a cloud provider's managed service. Finally, the customer can use a managed service to optimize their trained model for certain devices, e.g. edge devices, and deploy their model into production with governance and monitoring.

3.1 State-of-the-Art (SOTA)

3.1.1 What is the SOTA for Cloud-Native ML/DL workloads? In the space of Cloud-Native for ML/DL workloads there is no clear SOTA as each of three major cloud providers has their own specific advantages. It is clear however, that AWS is the leader in the space in terms of adoption as many enterprises had already leveraged AWS prior to their interests in ML/DL and have continued with that platform.

3.1.2 What features does a SOTA offering of Cloud-Native for ML/DL have? A SOTA Cloud-Native offering for ML/DL workloads would need to find the optimal balance between many dimensions that customers find important. These dimensions include: scalability,

Table 3: Pros and Cons of Cloud Native for ML/DL workloads.

	AWS SageMaker	GCP Vertex AI	Microsoft Azure ML
Pros	- Good for experienced users - Flexible and customizable - Wide array of services	- Support for both custom and AutoML - TPU support - Strong integrations (e.g. Kubeflow) - Meant for both novice and experienced users	- Good for novice users - Easy to use drag-and-drop UI - Vast array of out-of-the-box algorithms supported
Cons	- Code heavy - Difficult for novice users	- Still under development - Lack of documentation	- Meant for smaller scale jobs - Limited for experienced customers

cost efficiency, usability, customizability, flexibility, performance, and more. A SOTA offering would also need to excel in all of those dimensions for each stage of the ML/DL workflow. It would also be important for the offering to provide support for different types of workloads including real-time and batch workloads. A SOTA offering would also need to support the needs of both novice and expert customers since the offering must be accessible for all.

3.2 Deep Dive

3.2.1 AWS SageMaker. SageMaker [4, 14] was released in Fall 2017, and has continued to see widespread adoption since. AWS was the first cloud provider, and has benefited greatly from vendor lock-in as many customers who started with the platform have decided to forgo the benefits of other platforms in favor of not facing the challenges of migrating away from AWS. Hence, SageMaker became the go-to solution for current AWS customers that were interested in solutions for cloud-native ML/DL workloads. SageMaker has also steadily increased the number of managed cloud-native services for ML/DL workloads it provides well beyond the eight stage workflow outlined above. This is important for enterprises who have specific needs such as AWS Neo for edge model training. SageMaker is also very well documented and there exist many tutorials for how to get up and running quickly. Where SageMaker begins to differ from other cloud-native offerings for ML/DL workloads is its usage model. When using SageMaker, the customer mainly works inside of Jupyter Notebooks and relies heavily on code. The result of this is that customers who are comfortable with the details of ML/DL workloads are more suited for SageMaker whereas novice customers may have more trouble. This cloud-native service also exposes system-level details for an ML/DL workload to the customer including storage and compute resource details.

In summary, AWS SageMaker is great for customers in the following cases: the customer is already using AWS, the customer needs to run large-scale and custom ML/DL workloads, and/or the customer is comfortable working with Jupyter Notebooks and some system-level details. SageMaker also offers a wide variety of cloud-native services for specific types of ML/DL workloads which means that customers with niche requirements may benefit from these services.

3.2.2 GCP Vertex AI. Vertex AI [15, 25, 26], is fairly new as it was released in Spring 2021, but the components that it uses are not. Vertex AI's single unified interface is a combination of GCP's prior cloud native ML/DL services including GCP AI Platform, GCP Machine Learning Platform, and GCP AutoML. Since Vertex AI is a newer service, it has had the time to learn from and improve on the weaknesses of its older competitors such as AWS SageMaker. Vertex AI offers services for all of the eight stages of the ML/DL workflow. The major improvement that came from the release of GCP Vertex AI was its increased support for AutoML. Previously, GCP AI/ML's offerings were focused on providing tools and services that met the needs of data scientists and ML experts who needed to implement custom solutions for the ML/DL workflow. Now, novice customers with little to no ML/DL expertise can use the GCP AutoML workflow for their ML/DL workloads. Another strength of Vertex AI is it's integrations with other tools such as Kubeflow and TensorFlow. Finally, GCP provides a specialized hardware named the Tensor Processing Unit (TPU) which has become the desirable hardware solution for ML/DL workloads. Vertex AI has many strengths but it also has weaknesses. The documentation for the Vertex AI platform is lacking and can be difficult to navigate at times. This can cause unnecessary frustration and delays for customers. Another weakness is that many of the services offered by Vertex AI are still being iterated on and are not mature yet. A third weakness of the platform is that it is still not widely adopted due to its relatively late entry into the cloud-native AI/ML market.

In summary, GCP Vertex AI should be used in the following cases: when the customer desires or requires TPUs for their ML/DL workloads, when the customer wants flexibility to implement both custom and AutoML solutions within their organization, and/or when the customer wants to leverage Vertex AI's integrations with other services such as Kubernetes. Vertex AI is also a great choice for enterprises who are migrating their ML/DL workloads to the cloud for the first time as it has less of a learning curve and a more feature rich ecosystem than its competitors.

3.2.3 Microsoft Azure ML. Microsoft Azure ML [1, 11, 12, 23] was released in 2015 and has continued to improve since due to new feature releases such as Azure ML Studio. Even though Azure ML has been around for a longer time than its competitors, its competitive edge has come from Azure ML Studio. Azure ML Studio is a visual designer, i.e. drag-and-drop interface, for building little-to-no code models. It has a modular design and supports all of the stages for the ML/DL workflow. Azure ML also supports a vast amount of algorithms out of the box. All of these features have made Azure ML the go-to choice for customers with little to no experience with ML/DL workloads. An example of a customer who may want to use Azure ML is a business analyst that has little experience with the ML/DL workflow and small ML/DL workload. The ease-of-use

that Azure ML Studio provides comes at the cost of decreased flexibility. Hence, if customers require the increased flexibility they will need to move to the classic offering of Azure ML which requires proficiency in Jupyter Notebooks.

In summary, Microsoft Azure ML should be used in the following cases: when the customer is not well-versed in ML/DL workloads, when the customer wants a visual representation of their ML/DL workloads, and/or when the customer needs to perform a relatively small and simple job. Azure ML is a very powerful cloud native service for ML/DL workloads, however it is targeted towards a specific audience of novice customers which is different from its competitors.

3.3 Research Challenges

3.3.1 What are the common challenges faced by Cloud-Native for ML/DL workloads today? All of the cloud providers are working constantly to improve their cloud-native services for ML/DL workloads. The most common and important challenge that cloud providers face today, in terms of cloud-native services for ML/DL workloads, is on-boarding new customers who have little-to-no experience with ML/DL but require ML/DL solutions. In past years, cloud-native ML/DL workloads were mainly handled by experts and therefore all cloud-native services for ML/DL workloads were geared towards experts. Today this is no longer the case and the challenge of supporting these new types of customers is not trivial.

3.3.2 Where is the field going? The future of cloud-native services for ML/DL workloads is bright. The introduction and growth of AutoML will continue as more enterprises will opt for the MLaaS model rather than going through expensive talent acquisitions. Hybrid cloud solutions are another important aspect of cloud-native services for ML/DL workloads since customers may have already invested heavily in physical resources, but want to leverage the cloud provider's services for ML/DL. A third interesting direction that the field is going in is support for multi-cloud solutions. Customers are beginning to no longer be complacent with being locked into a single cloud provider and may also want to leverage different cloud provider's cloud-services for different parts of their ML/DL workload. A trend that has already begun, and will continue in the future, is support for edge and IoT devices since advances in hardware are making ML/DL in those use cases plausible.

4 CONCLUSION

This work performs an extensive survey on cloud-native and serverless frameworks for ML/DL workloads. The results of our survey indicate that both the serverless and cloud-native frameworks for ML/DL workloads have great benefits and face interesting challenges. Serverless for ML/DL workloads is still mainly a research direction that is focused on optimizing the execution and management of ML/DL workloads. FaaS for ML/DL workloads is feasible for certain types of ML/DL workloads today and future improvements in the field will increase its generalizability to a wider range of workloads. Cloud-native for ML/DL workloads is mainly focused on supporting enterprises with wide ranges of ML/DL expertise in achieving their ML/DL goals. Each cloud provider offers cloud-native services with different strengths, hence the cloud-native service used for a customer's ML/DL workload must be chosen based upon multiple factors including cost, expertise, size of the project, and performance.

ACKNOWLEDGMENTS

Thank you to Professor Arun Kumar for the support and guidance throughout the process of creating this work.

REFERENCES

[1] altexsoft. 2021. *Comparing Machine Learning as a Service: Amazon, Microsoft Azure, Google Cloud AI, IBM Watson.* Retrieved December 2nd, 2021 from https://www.altexsoft.com/blog/datascience/comparing-machine-learning-as-a-service-amazon-microsoft-azure-google-cloud-ai-ibm-watson/

[2] Amazon. [n. d.]. *Amazon Web Services.* Retrieved December 2nd, 2021 from https://aws.amazon.com

[3] Amazon. [n. d.]. *AWS Lambda.* Retrieved December 2nd, 2021 from https://aws.amazon.com/lambda/

[4] Amazon. [n. d.]. *AWS SageMaker.* Retrieved December 2nd, 2021 from https://aws.amazon.com/sagemaker/

[5] Apache. [n. d.]. *Hadoop.* Retrieved December 2nd, 2021 from https://hadoop.apache.org

[6] Apache. [n. d.]. *Spark.* Retrieved December 2nd, 2021 from https://spark.apache.org

[7] Azure. [n. d.]. *Microsoft Azure Functions.* Retrieved December 2nd, 2021 from https://azure.microsoft.com/en-us/services/functions/

[8] Tobias Bohnhoff. 2019. *Machine Learning as a Service — The Top Cloud Platform and AI Vendors.* Retrieved December 2nd, 2021 from https://medium.com/appanion/machine-learning-as-a-service-the-top-cloud-platform-and-ai-vendors-2df45d51374d

[9] Joao Carreira, Pedro Fonseca, Alexey Tumanov, Andrew Zhang, and Randy Katz. 2019. Cirrus: A Serverless Framework for End-to-End ML Workflows. *Proceedings of the ACM Symposium on Cloud Computing* (2019), 12 pages. https://doi.org/10.1145/3357223.3362711

[10] Alex Chung. 2021. *Comparing Cloud MLOps platforms, From a former AWS SageMaker PM.* Retrieved December 2nd, 2021 from https://towardsdatascience.com/comparing-cloud-mlops-platform-from-a-former-aws-sagemaker-pm-115ced28239b

[11] Steve Dille. 2019. *How to Decide Between Amazon SageMaker and Microsoft Azure Machine Learning Studio.* Retrieved December 2nd, 2021 from https://towardsdatascience.com/how-to-decide-between-amazon-sagemaker-and-microsoft-azure-machine-learning-studio-157a08af839a

[12] erbis. 2021. *BUILDING A MACHINE LEARNING PROJECT: AWS VS MICROSOFT AZURE.* Retrieved December 2nd, 2021 from https://erbis.com/blog/build-ml-model-aws-vs-azure/

[13] Abadi et al. 2016. TensorFlow: A System for Large-Scale Machine Learning. *Proceedings of the 12th USENIX Symposium on Operating Systems Design and Implementation (OSDI '16)* (Nov 2016).

[14] Liberty et al. 2020. Elastic Machine Learning Algorithms in Amazon SageMaker. *Proceedings of the 2020 ACM SIGMOD Inter- national Conference on Management of Data (SIGMOD'20)* (June 2020). https://doi.org/10.1145/3318464.3386126

[15] Google. [n. d.]. *GCP Vertex AI.* Retrieved December 2nd, 2021 from https://cloud.google.com/vertex-ai/docs/beginner/beginners-guide

[16] Google. [n. d.]. *Google Cloud Functions.* Retrieved December 2nd, 2021 from https://cloud.google.com/functions

[17] Google. [n. d.]. *Google Cloud Platform.* Retrieved December 2nd, 2021 from https://cloud.google.com

[18] Jiawei Jiang, Shaoduo Gan, Yue Liu, Fanlin Wang, Gustavo Alonso, Ana Klimovic, Ankit Singla, Wentao Wu, and Ce Zhang. [n. d.]. Towards Demystifying Serverless Machine Learning Training. ([n. d.]).

[19] Alexey Tumanov Andrew Zhang Joao Carreira, Pedro Fonseca and Randy Katz. 2018. A case for serverless machine learning. *Workshop on Systems for ML and Open Source Software at NeurIPS* (2018). https://www.cs.purdue.edu/homes/pfonseca/papers/mlsys18-disaggregatedml.pdf

[20] Eric Jonas, Qifan Pu, Shivaram Venkataraman, Ion Stoica, and Benjamin Recht. 2017. Occupy the Cloud: Distributed Computing for the 99%. *Proceedings of the 2017 Symposium on Cloud Computing* (2017), 7 pages. https://doi.org/10.1145/3127479.3128601

[21] Dilma Da Silva Lang Feng, Prabhakar Kudva and Jiang Hu. 2018. Exploring serverless computing for neural network training. *2018 IEEE 11th International Conference on Cloud Computing (CLOUD)* (2018). https://par.nsf.gov/servlets/purl/10110813

[22] Microsoft. [n. d.]. *Microsoft Azure.* Retrieved December 2nd, 2021 from https://azure.microsoft.com/en-us/

[23] Microsoft. [n. d.]. *Microsoft Azure ML.* Retrieved December 2nd, 2021 from https://azure.microsoft.com/en-us/services/machine-learning/#product-overview

[24] Microsoft. 2021. *What is Cloud Native?* Retrieved December 2nd, 2021 from https://docs.microsoft.com/en-us/dotnet/architecture/cloud-native/definition
[25] Janakiram MSV. 2021. *Google Revamps Its Cloud-Based Machine Learning Platform Yet Again.* Retrieved December 2nd, 2021 from https://www.forbes.com/sites/janakirammsv/2021/05/19/google-revamps-its-cloud-based-machine-learning-platform-yet-again/?sh=1eea75d765ac
[26] Matt Squire. 2021. *Vertex AI - does it live up to the MLOps hype?* Retrieved December 2nd, 2021 from https://fuzzylabs.ai/blog/vertex-ai-the-hype/
[27] Thomas Courtade Michael W Mahoney Vipul Gupta, Swanand Kadhe and Kannan Ramchandran. 2019. Oversketched newton: Fast convex optimization for serverless systems. (2019). https://arxiv.org/pdf/1903.08857.pdf
[28] Hao Wang, Di Niu, and Baochun Li. 2019. Distributed Machine Learning with a Serverless Architecture. *IEEE INFOCOM 2019 - IEEE Conference on Computer Communications* (2019). https://ieeexplore.ieee.org/stamp/stamp.jsp?arnumber=8737391