2018 First International Conference on Artificial Intelligence for Industries

Kubebench: A Benchmarking Platform for ML Workloads

Xinyuan Huang†, Amit Kumar Saha†, Debojyoti Dutta†, Ce Gao*‡
†Cisco Systems, *Caicloud, ‡Shanghai Jiao Tong University

***Abstract*—Machine Learning (ML) workloads are becoming mainstream in the enterprise but the plethora of choices around ML toolkits and multi-cloud infrastructure make it difficult to compare their performance and costs. In this paper, we motivate the need for benchmarking ML systems in a consistent way, discuss the requirements of an ML benchmarking platform, and propose a design that satisfies the requirements. We present Kubebench, an example open-source implementation of an ML benchmarking platform based on Kubeflow, itself an open-source project for managing any ML stack on Kubernetes, a widely used container management platform.**

***Index Terms*—machine-learning; benchmarking; kubebench; kubeflow; kubernetes**

I. Introduction

Machine Learning (ML) systems are of strategic importance for many enterprises and is increasingly influencing business decisions. The field is progressing fast and new improvements are being frequently published (peer-reviewed or otherwise), almost weekly. Recent famous results, more often than not, are based on extremely large data sets and use up a lot of computational power in the form of CPUs, GPUs, and TPUs [1]. What is usually not obvious is the *systems engineering issues* that are required to make such ML systems work reliably and at scale, and in a multi-cloud world that enterprises have to deal with. Even though the ML algorithms are mostly public and often have open source implementations with active community engagement, the real hurdle in *democratizing* machine learning and putting it to production are the engineering issues in reliably scaling any ML solution, during both the training and the inference stages. Also in a multi-cloud environment, one needs to manage public cloud costs and operational costs for on-premises infrastructure.

The open-source Kubeflow [2] project is one of the first serious open-source attempts to democratize ML on container platforms, and "*is dedicated to making deployments of machine learning (ML) workflows on Kubernetes simple, portable and scalable.*" Just as the Kubernetes [3] project has become a popular choice for deploying, scaling, and managing containerized applications, Kubeflow is an attempt to reducing the barrier to entry in deploying, scaling, and managing ML solutions.

Kubeflow is simply a ML lifecycle manager that helps run workflows on any Kubernetes cluster. This decouples the responsibilities of a data scientist and a ML *devsecops* engineer. It enables the ML community to focus on just the ML models and algorithms while delegating the actual deployment, lifecycle management, and scalability of the end application to the Kubernetes ecosystem comprising of Kuberetes, Kubeflow, and service meshes like Istio [4]. Thus, the data scientist is left free to develop models in the framework of his/her choice, move it to Kubeflow and automatically be able to run the application on on-premise infrastructure, and then, by simply pointing to a Kubernetes cluster that is running on the cloud (say, Amazon's ECS [5] or Google's GKE [6]), can get the application to run with the scale and reliability of a production-grade, cloud-based Kubernetes cluster. Kubeflow started with the support of only TensorFlow [7] jobs but has already started supporting PyTorch [8] jobs.

Enterprise are confronted with an array of complex choices to make in order to put ML in production. For example, these decisions have to make sense in the multi-cloud world we live in. Some of the following choices are around performance and cost optimization that each enterprise has to make for their specific use cases:

- There are several popular ML toolkits such as TensorFlow [7], PyTorch [8], and Caffe [9]. Without trying out all of these, it is impossible to say which toolkit will perform the best for any specific use case.
- It is impossible to know how an ML stack will perform for a given ML workload if the underlying infrastructure platform specifications were to be changed. For example, adding more machines (or containers or VMs) might actually make the overall system slower, as shown by Zhang et al. [10].

The *contributions* of this paper are two fold. *First*, the paper enumerates the design requirements of benchmarking platforms and infrastructure, to address the above mentioned issues and *second*, the paper presents the design of Kubebench that satisfies the design requirements. Kubebench is an open-source ML benchmarking platform based on Kubeflow. To the best of our knowledge, this is the first paper that addresses the above problems. This paper focuses on the ML infra as opposed to defining ML workloads, which is beyond the scope of this paper.

The rest of this paper is organized as follows. We explain the benchmarking requirements in Section II, following which Section III has the design details of the Kubebench benchmarking platform. We present related work in Section IV and finally we conclude in Section V.

II. Benchmarking Requirements

The requirements presented here are generic and independent of the Kubeflow project. Though the design proposal

978-1-5386-9209-7/18/$31.00 ©2018 IEEE
DOI 10.1109/ai4i.2018.00025

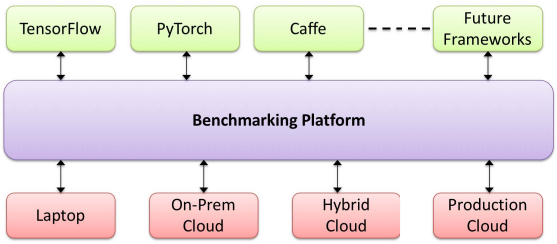

Fig. 1. Schematic of the suggested benchmarking platform.

presented next in Section III is based on Kubeflow, other implementations satisfying these requirements are very much possible. The following should be the high level requirements for a benchmarking platform, in no particular order (Fig. 1 shows a high level schematic):

- *Support for multiple ML frameworks*. Since there are multiple popular ML frameworks, such as TensorFlow [7] and PyTorch [8], a benchmarking platform must support multiple frameworks. This ensures that the platform can be used without any vendor lock-in. Implicit in this requirement is the additional requirement that the benchmarking platform should easily be able to support future ML frameworks.
- *Portable.* One of the most important requirements for a benchmarking platform is the ability to easily move from one environment to another. For ML applications, it is quite common that the ML expert runs experiments on models on his/her own laptop to get some preliminary performance numbers, then moves to an on-premise infrastructure to test a larger version (on a larger data set), and finally, when satisfied with the performance of the model, moves to a production-grade cloud to test a scaled out version. Of course, each of these setups can have a mix of different computation elements such as, CPUs, GPUs, and TPUs. The benchmarking platform should allow seamless movement from one infrastructure to another, possibly by changing some human readable configuration and without having to change anything in the underlying ML training/serving job. However, if the production grade, cloud-scale system needs distributed training whereas the smaller scale, laptop or on-premise versions, uses a single training job, then there would have to be some changes made.
- *Extreme scalability.* Since realistic ML applications usually run with very large data sets, the model training is also done in a scaled out distributed system. Consequently, the benchmarking platform should be able to run at production-scale, thus allowing quantitative evaluation and comparison of realistic deployment scenarios.
- *Flexible and extensible API.* The performance of a distributed ML job is affected by many factors including, model parameters, distribution scale, scheduling policy, accelerator settings, network configurations, etc. The benchmark platform must have a flexible API so that the user can easily specify variables of interest corresponding to each specific benchmark case. It is also important to have the API extensible so that new kinds of model parameters and system configurations can be added on the fly as the requirement grows with new benchmark scenarios.
- *Based on open-source*. Last, but certainly not the least, the benchmarking platform should be open-source so that it can be used by everyone without any financial liability. This paper will not argue for the advantages of open-source software but the fact that open-source software, among other things, *is customizable and flexible*, *avoids vendor lock-in*, and probably most importantly, *can be audited*, should be reason enough.

In addition to the above mentioned requirements, it is also important to identify *non-goals* so that the benchmarking platform does not get overloaded with non-essential features. The platform by itself should not be involved with making the results persistent or to provide any sort of graphical user interface for running, monitoring, or reporting and visualizing results. These features, albeit useful, should be done via a separate software entity. *The primary goal of the benchmarking platform is to allow the users to run ML workloads using different ML frameworks and on different infrastructure*. Additional community efforts similar to TPC [11] benchmarks would be needed to consolidate and publish benchmark results from multiple vendors. Already such efforts are underway in the form of MLPerf [12] and understandably it is emulating the TPC process wherever possible.

III. Design of Kubebench

In this section we describe the design of Kubebench, a benchmarking platform based on Kubeflow that runs benchmark jobs on Kubernetes. As already mentioned, this is an example implementation and other implementations satisfying the requirements specified in Section II are very much possible.

A. Workflow Design

Kubebench defines a workflow that leverages Kubeflow to deploy ML workloads and implements a set of steps to augment the ML workloads with configuration and result processing functionalities. In this paper, we consider two kinds of jobs in this process. Namely, Kubebench job and Kubeflow job, where *a Kubebench job defines the workflow of a benchmark task*, while *a Kubeflow job defines the workloads of the benchmark*. The major steps in running a Kubebench job, shown in Fig. 2, are as follows:

1) *Configuration*. The first step is to setup a benchmark job configuration. This configuration contains various settings, including *system settings* such as cluster sizes, workload images, accelerator configurations, etc. and *model parameters* such as model type and batch size. The configuration consists of two parts: a *template*[1] that

[1]In Kubernetes, such templates are typically packaged in Ksonnet Packages or Helm Charts.

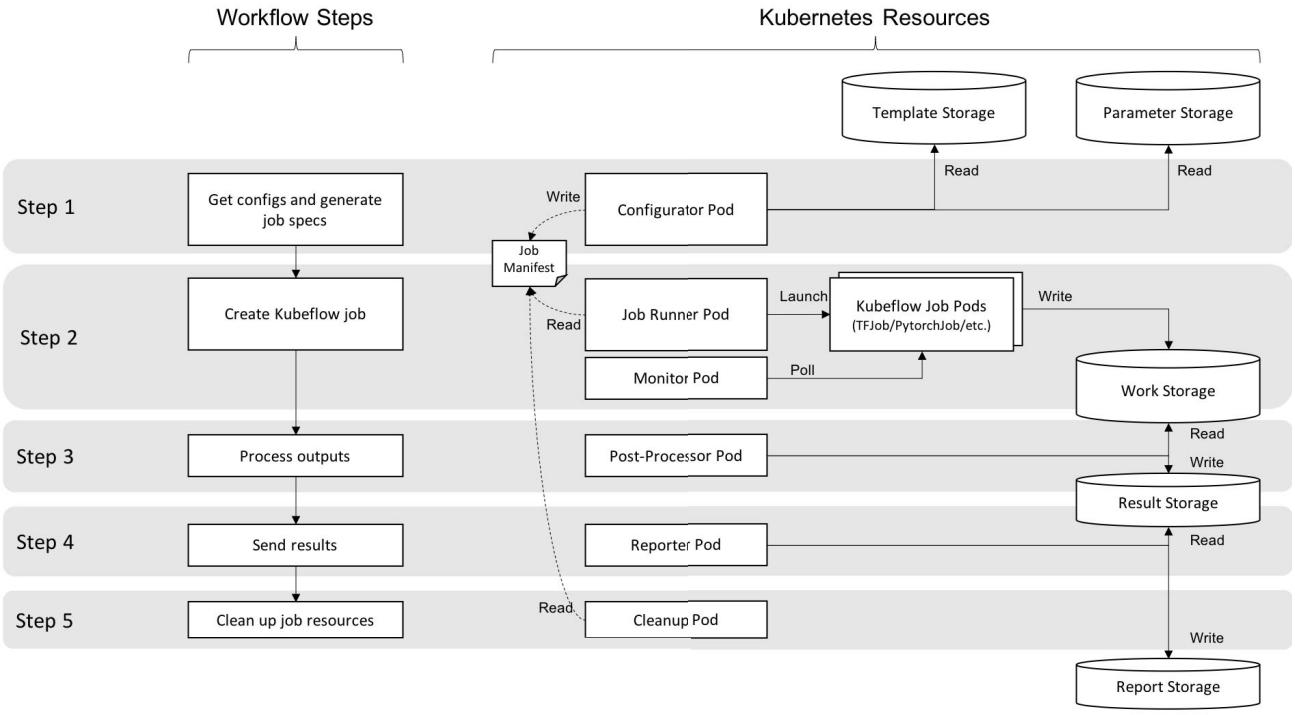

Fig. 2. Workflow of the Kubebench benchmarking platform running on Google Kubernetes Engine (GKE). The configuration and the results can be either stored in a database (DB) or PVC, which stands for Persistent Volume Claim, a standard way for requesting and claiming a persistent resource in GKE.

defines the overall job specifications with configurable variables, and a set of parameters that specify variable values. The configuration is parsed to produce a job *manifest*[2] that defines a Kubeflow job. Since Kubeflow runs on an underlying Kubernetes cluster, this step is implemented in a *pod*[3].

2) *Run Kubeflow Job*. In this step, Kubeflow runs the job produced in the previous step. A runner pod reads the job manifest and launches Kubeflow job pods based on the system settings and model parameters specified in the manifest. At the same time, a monitor pod keeps track of the job status. The container images and corresponding source codes used to run the job can be derived from pre-existing benchmark tools. Kubebench is oblivious to the underlying implementation of these images.
3) *Compile Outputs*. Once the benchmark job completes, a post-processor parses the logs and output from the completed jobs to produce a result report. This report, along with other useful information (e.g. aggregated logs, etc.), are persisted in a result storage.
4) *Report Results*. A reporter pod reads in the report information of the single benchmark experiment, and merges it with the aggregated report of all experiments in a user-specified external storage.
5) *Clean Up*. Finally, all the job resources created in the previous steps are deleted so that the system is left in the same state that it was before the benchmark workflow started.

B. API Design

The Kubebench configuration consists of two tiers of parameters. The *first tier* parameters control the behavior of the Kubebench workflow, where container images and corresponding arguments are specified for the Configurator, Post-Processor, and Reporter pods (shown in Fig. 2 and described in Section III-A). Other parameters, such as persistent data storages and their access credentials, are also specified alongside. The *second tier* configuration, as shown in Table I, controls the behaviors of the Kubeflow job, which mainly consists of a reference to a *Ksonnet prototype* [4] for the Kubeflow job, and a list of parameters corresponding to the prototype. The 2-tier configuration structure decouples the benchmark workflow from the ML workloads, thus allows reusing a workflow without making frequent changes for a big batch of different ML workload scenarios.

The Kubebench platform satisfies the requirements described in Section II. It *supports multiple ML frameworks*

[2]A Kubernetes manifest defines the specifications of Kubernetes resources.

[3]A Kubernetes pod is a group of containers that are deployed together on the same host. In several practical scenarios single containers are used and so "pod" and "container" are sometimes changed interchangeably.

[4]Ksonnet is a framework for defining and deploying Kubernetes applications. A Ksonnet prototype is a boilerplate of the application with configurable parameters.

Paremeter Name	Example	Explanation
name	example-benchmark-job	Name for easier identification
namespace	example-benchmark	Kubernetes namespace for isolation
image	benchmark-img-cpu:v1	Container image for running ML workload
imageGpu	benchmark-img-gpu:v1	GPU version of "image"
args	"−−batch_size=100,−−num_batches=32,−−model=resnet50"	Command line arguments to "image"
numMasters	1	Number of masters in Kubeflow
numPs	2	Number of parameter servers in Kubeflow
numWorkers	4	Number of workers in Kubeflow
numGpus	4	Number of GPUs to be used by Kubeflow

TABLE I
EXAMPLE 2ND-TIER CONFIGURATION OF KUBEBENCH. SOME OF THESE PARAMETERS ARE KUBEFLOW SPECIFIC AND A DETAILED EXPLANATION OF SUCH PARAMETERS IS OUTSIDE THE SCOPE OF THIS PAPER.

by leveraging the underlying Kubeflow's ability to run ML jobs of multiple ML frameworks. It is *portable* because it simply requires a Kubernetes cluster to run the job; the cluster can be on one's own laptop, on on-premise clouds, on hybrid clouds, or on production-grade public clouds. It has a *flexible and extensible API* and since Kubernetes clusters can scale to extremely large sizes, Kubebench is *extremely scalable* as well. Finally of course, Kubebench is *open-source*.

The implementation of this design is currently under active development and readers are welcome to look at *and contribute to* the open-source Kubebench [13] project.

IV. RELATED WORK

There are projects similar to Kubeflow such as Intel's Machine Learning container Templates (MLT) [14] and IBM's Fabric for Deep Learning (FfDL) [15]. However, to the best of our knowledge, there is no general purpose benchmarking platforms for machine learning workloads for the container ecosystem. There are benchmark results and sample benchmark code for individual ML frameworks [16], [17] but these are usually meant for specific setups and are not easily portable, especially to a containerized platform.

The MLPerf [12] project has reference implementations for different ML frameworks but there is no standard machinery to run these on Kubernetes clusters (though they do run the benchmarks as containers). We expect a lot of synergy between Kubebench and MLPerf since Kubebench can leverage the MLPerf reference implementations and benchmark them on Kubernetes clusters of any size (and of course, report the benchmark results back to MLPerF).

V. CONCLUSION

This paper presents a benchmarking platform for machine learning workloads, especially for enterprises embracing a multi-cloud container world. Additionally, the paper also presents the design and implementation of Kubebench, an open-source benchmarking platform for running ML benchmark tests using Kubeflow, an open-source project for the deployment and the lifecycle management of ML workflows on Kubernetes. This allows enterprises to leverage quantitative data to make educated decisions about models, the toolkits in which the models are implemented, and the underlying multi-cloud infrastructure for running these. Kubebench is under active development in the open-source community within the Kubeflow effort.

Note that we do not present new reference workloads and we leverage standard workloads from MLPerf or Tensorflow. This platform is a consistent way to describe, run, and collect metrics for the ML lifecycle, given a reference workload. We hope that this paper motivates further discussions and collaborations around benchmarking machine learning workloads.

REFERENCES

[1] Tensor Processing Unit. [Online]. Available: https://cloud.google.com/tpu/docs/tpus

[2] Kubeflow: Machine Learning Toolkit for Kubernetes. [Online]. Available: https://github.com/kubeflow/kubeflow

[3] Kubernetes: Production-Grade Container Orchestration. [Online]. Available: https://kubernetes.io/

[4] Istio: An open platform to connect, manage, and secure microservices. [Online]. Available: https://istio.io/

[5] Amazon Elastic Container Service. [Online]. Available: https://aws.amazon.com/ecs/

[6] Google Kubernetes Engine. [Online]. Available: https://cloud.google.com/kubernetes-engine/

[7] TensorFlow: An open source machine learning framework for everyone. [Online]. Available: https://www.tensorflow.org/

[8] PyTorch: a deep learning framework for fast, flexible experimentation. [Online]. Available: https://pytorch.org/

[9] Caffe: a fast open framework for deep learning. [Online]. Available: http://caffe.berkeleyvision.org/

[10] H. Zhang, Z. Zheng, S. Xu, W. Dai, Q. Ho, X. Liang, Z. Hu, J. Wei, P. Xie, and E. P. Xing, "Poseidon: An efficient communication architecture for distributed deep learning on GPU clusters," in *2017 USENIX Annual Technical Conference (USENIX ATC 17)*. Santa Clara, CA: USENIX Association, 2017, pp. 181–193. [Online]. Available: https://www.usenix.org/conference/atc17/technical-sessions/presentation/zhang

[11] TPC Benchmarks. [Online]. Available: http://www.tpc.org/information/benchmarks.asp

[12] MLPerf: A broad ML benchmark suite for measuring performance of ML software frameworks, ML hardware accelerator, and ML cloud platforms. [Online]. Available: https://mlperf.org/

[13] Kubebench: A Benchmarking Platform for Kubeflow. [Online]. Available: https://github.com/kubeflow/kubebench

[14] Machine Learning Container Templates. [Online]. Available: https://github.com/IntelAI/mlt

[15] Fabric for Deep Learning. [Online]. Available: https://github.com/IBM/FfDL

[16] TensorFlow Benchmarks. [Online]. Available: https://www.tensorflow.org/performance/benchmarks

[17] PyTorch Benchmark Code. [Online]. Available: https://github.com/pytorch/benchmark