Comparative analysis of distributed data parallel training packages

Andreea Zaharia — R244 Mini-project proposal

Motivation

Contribution of the project

- Deep learning models
- Distributed training across multiple workers/multiple GPUs
- Multiple packages implementing similar solutions
- Different optimisations or communication backends
- Lack of performance comparison between the most used frameworks

Open-source solutions

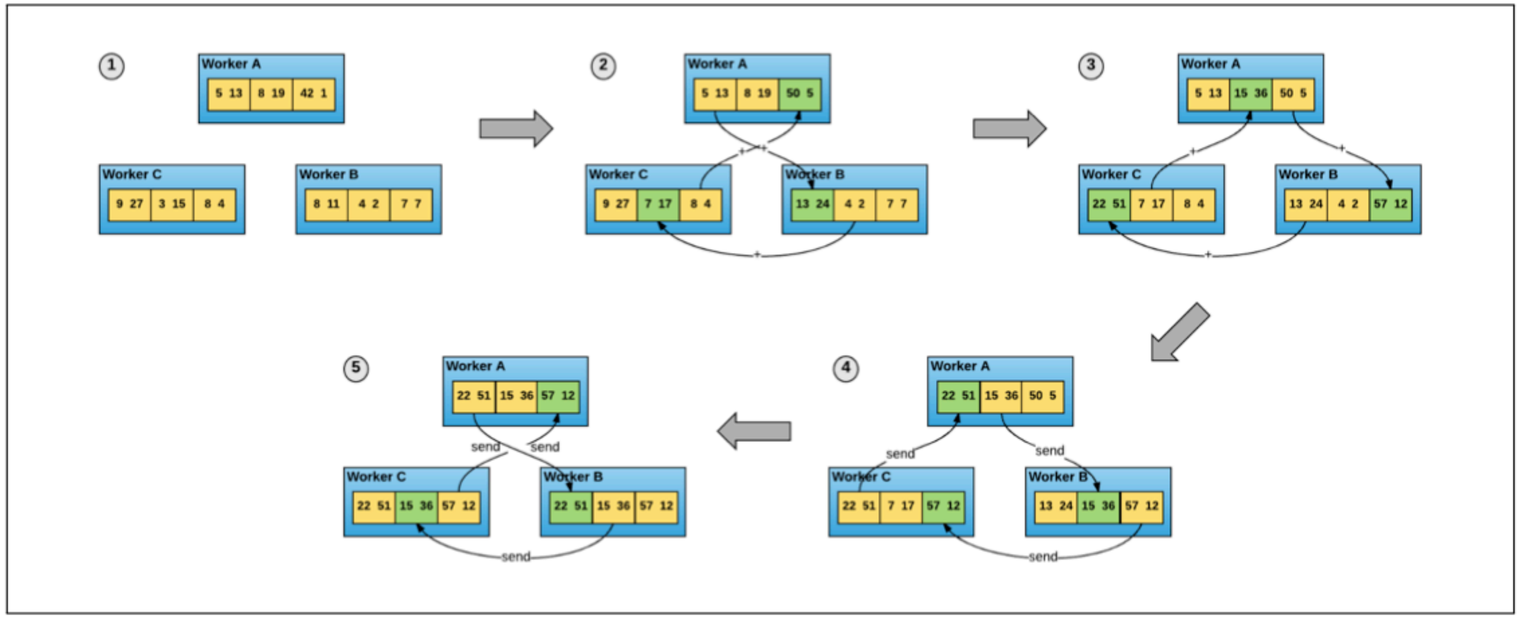
Native all-reduce modules

- Synchronous, decentralised approach
- All-Reduce pattern
- Overlap between communication and computation
- Native to their respective frameworks:
  - PyTorch DistributedDataParallel and DataParallel
  - Tensorflow MirroredStrategy and MultiWorkerMirroredStrategy

Open-source solutions

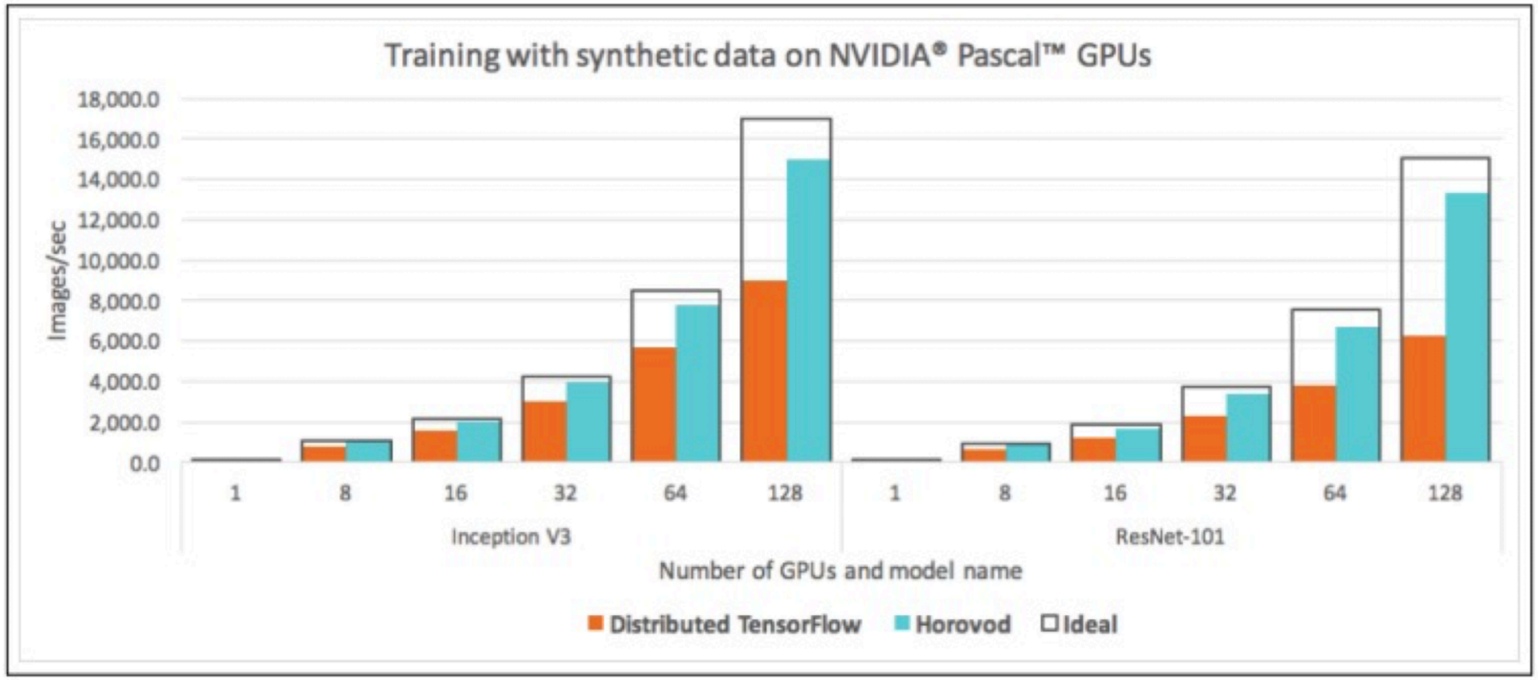
Horovod

- Supports Tensorflow, PyTorch, and MXNet.
- Ring all-reduce.



Horovod performance

Actual throughput: Tf vs Horovod vs ideal



arXiv:1802.05799v3 [cs.LG] 21 Feb 2018

Horovod: fast and easy distributed deep learning in TensorFlow

Alexander Sergeev
Uber Technologies, Inc.
asergeev@uber.com

Mike Del Balso
Uber Technologies, Inc.
mdb@uber.com

Abstract

Training modern deep learning models requires large amounts of computation, often provided by GPUs. Scaling computation from one GPU to many can enable much faster training and research progress but entails two complications. First, the training library must support inter-GPU communication. Depending on the particular methods employed, this communication may entail anywhere from negligible to significant overhead. Second, the user must modify his or her training code to take advantage of inter-GPU communication. Depending on the training library's API, the modification required may be either significant or minimal.

Existing methods for enabling multi-GPU training under the TensorFlow library entail non-negligible communication overhead and require users to heavily modify their model-building code, leading many researchers to avoid the whole mess and stick with slower single-GPU training. In this paper we introduce Horovod, an open source library that improves on both obstructions to scaling: it employs efficient inter-GPU communication via ring reduction and requires only a few lines of modification to user code, enabling faster, easier distributed training in TensorFlow. Horovod is available under the Apache 2.0 license at `https://github.com/uber/horovod`.

1 Introduction

Over the past few years, advances in deep learning have driven tremendous progress in image processing, speech recognition, and forecasting. At Uber, we apply deep learning across our business; from self-driving research to trip forecasting and fraud prevention, deep learning enables our engineers and data scientists to create better experiences for our users.

TensorFlow [1] has become a preferred deep learning library at Uber for a variety of reasons. To start, the framework is one of the most widely used open source frameworks for deep learning, which makes it easy to onboard new users. It also combines high performance with an ability to tinker with low-level model details—for instance, we can use both high-level APIs, such as Keras [2], and implement our own custom operators using NVIDIA's CUDA toolkit. Additionally, TensorFlow has end-to-end support for a wide variety of deep learning use cases, from conducting exploratory research to deploying models in production on cloud servers, mobile apps, and even self-driving vehicles.

In September 2017, Uber Engineering introduced Michelangelo [3], an internal ML-as-a-service platform that democratizes machine learning and makes it easy to build and deploy these systems at scale. In this paper, we introduce Horovod, an open-source component of Michelangelo's deep learning toolkit which makes it easier to start—and speed up—distributed deep learning projects with TensorFlow. Horovod is available under the Apache 2.0 license at `https://github.com/uber/horovod`.

Figure 6: A comparison of images processed per second with standard distributed TensorFlow and Horovod when running a distributed training job over different numbers of NVIDIA Pascal GPUs for Inception V3 and ResNet-101 TensorFlow models over 25GbE TCP.

Experiment ideas

Experiment design and implementation

- Implement the same models in Tensorflow and PyTorch
  - Training tasks: e.g. ResNet50, MobileNetV2, AlexNet
  - Multiple physical GPUs (prototyping on multiple logical GPUs)

- Run each with its native framework
- Run each with Horovod, instead of the native frameworks
- Vary communication backends (NCCL, GLOO).

Experiment ideas

Experiment metrics

- Comparative analysis of time and performance with respect to resources

- **Example metrics:**
  - Per-epoch accuracy as the number of GPUs increases
  - Speedup as the number of GPUs increases
  - Distributed train time per step (ms/step) and train time in total
  - Etc.

Objectives

Outcome and contribution

- Run the experiments presented (or equivalent ones)
- Compile the data into a performance evaluation
- Write a compelling comparative analysis of the three solutions
- Hypothesise on the causes of differences observed (if any)

Work plan

Preliminary reading and next steps

- Read and reviewed the PyTorch DDP paper.
- Read through existing evaluations of each framework.
- Explored options for setting up the multi-GPU system.

- **Next step:**
  - Implement the experiment code on the logical GPUs setup.
  - Configure an adjustable distributed system with feedback loop.

Questions? Suggestions?