A Scalable, High-Performance, and Fault-Tolerant Network Architecture for Distributed Machine Learning

Songtao Wang, Dan Li, Yang Cheng, Jinkun Geng, *Graduate Student Member, IEEE*, Yanshu Wang, Shuai Wang, Shutao Xia, and Jianping Wu, *Fellow, IEEE*

***Abstract*—In large-scale distributed machine learning (DML), the network performance between machines significantly impacts the speed of iterative training. In this paper we propose *BML*, a scalable, high-performance and fault-tolerant DML network architecture on top of Ethernet and commodity devices. BML builds on BCube topology, and runs a fully-distributed gradient synchronization algorithm. Compared to a Fat-Tree network with the same size, a BML network is expected to take much less time for gradient synchronization, for both low theoretical synchronization time and its benefit to RDMA transport. With server/link failures, the performance of BML degrades in a graceful way. Experiments of MNIST and VGG-19 benchmarks on a testbed with 9 dual-GPU servers show that, BML reduces the job completion time of DML training by up to 56.4% compared with Fat-Tree running state-of-the-art gradient synchronization algorithm.**

***Index Terms*—Distributed machine learning, gradient synchronization time, scalability, fault-tolerant.**

I. Introduction

MACHINE learning (ML) has become a core service in large companies [18]. The scale of modern ML training can be huge [5], [9], [21]. From our survey of a large internet company, a CTR (click through rate) estimation task trains a model of >100 billion features with >1PB training data. Given the memory size and processing capability of today's commodity machines, it is inevitable to run distributed machine learning (DML) on multiple machines. For instance, the internet company under survey currently uses several hundreds of dedicated machines to carry out the training for CTR estimation. With the ever-increasing training data and model sizes, it is expected that even larger-scale DML will appear in the near future.

Manuscript received July 24, 2018; revised April 7, 2019 and December 23, 2019; accepted April 30, 2020; approved by IEEE/ACM TRANSACTIONS ON NETWORKING Editor M. Andrews. Date of publication June 19, 2020; date of current version August 18, 2020. This work was supported in part by the National Key Research and Development Program of China under Grant 2019YFB1802600, in part by the Research and Development Program in Key Areas of Guangdong Province under Grant 2018B010113001, and in part by the National Natural Science Foundation of China under Grant 61772305 and Grant 61672499. Some preliminary results were published in Neurips 2018. In this journal article, we add the fault tolerance part, add more experiments and make the work more complete. *(Corresponding author: Dan Li.)*

Songtao Wang, Dan Li, Yang Cheng, Jinkun Geng, Yanshu Wang, Shuai Wang, and Jianping Wu are with the Department of Computer Science and Technology, Tsinghua University, Beijing 100084, China (e-mail: tolidan@mail.tsinghua.edu.cn).

Shutao Xia is with Tsinghua Shenzhen International Graduate School, Tsinghua University, Shenzhen 518055, China.

Digital Object Identifier 10.1109/TNET.2020.2999377

A typical ML training task trains a model iteratively until the parameters converge. In the widely-used *gradient descent* optimization method, in each iteration the algorithm uses a *minibatch* of training data to compute a *gradient*, which decides the changes to make to the parameters trained by the previous iteration. In DML, every machine iteratively trains a *sub-minibatch* (sub-minibatch means breaking a minibatch of input data into multiple pieces and each machine uses one piece.) of data and synchronizes the gradients with other machines. Ideally, more machines help reduce the training time. However, it has been shown that, when more machines are used in DML, we have to set a *smaller sub-minibatch size* per machine, so as to keep the aggregated minibatch over all the machines with a reasonable size. Otherwise, the large aggregated minibatch may cause the training to quickly converge to a worse model. For instance, a recent work from Facebook discloses that their translation service cannot currently train on large minibatches without degrading model quality [18].

A side effect of smaller sub-minibatch size per machine in larger-scale DML is the break of computation/communication balance. For example, an experiment from Amazon shows that [27], if the batch size on a GPU is 16, the processing time per batch stays stable from 1 GPU to 128 GPUs; while if the batch size on a GPU is 2, the processing time per batch under 128 GPUs increases by more than 6 times compared with the time per batch under a single GPU, because of the dominating communication cost. Therefore, in order to run DML in large scale, we need to carefully design the network with minimized communication overhead among machines.

We have the following requirements for a DML network. First, the network can extend to *large scale* based on *commodity devices*. In particular, IP/Ethernet is assumed as the underlying network protocol suite. Second, the network can achieve as low *gradient synchronization time (GST)* as possible. The GST is determined by the topology characteristic, gradient synchronization algorithm as well as the cost of transport protocols. Third, the network should be *fault-tolerant*. With server/link failures, we expect the training performance *gracefully* degrades, instead of steeply degrades or even crashes. Existing DML network architectures cannot meet all the requirements above. The Fat-Tree network [7] running state-of-the-art parameter server (PS) algorithm has