PURDUE UNIVERSITY | Department of Computer Science

PurNET NVIDIA vmware

Ultima: Robust and Tail-Optimal All-Reduce for Distributed Deep Learning

Ertza Warraich, Leonard Liu, Omer Shabtai, Yonatan Piasetzky, Shay Vargaftik,
Matty Kadosh, Lalith Suresh, and Muhammad Shahbaz

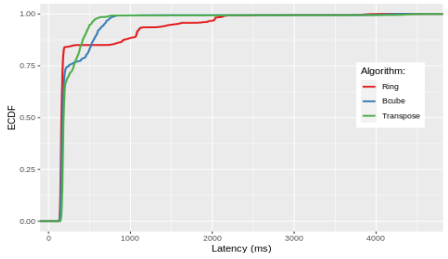
1. Abstract

- **Distributed data-parallel** (DDP) is the de-facto standard for **training** large-scale **deep-learning.** Hence, extensive efforts are put forth in reducing the *computation* as well as the *communication* time of a DDP job - but these efforts focus on reducing the **average job completion times** – making the system still **susceptible to long-tails**.
- We present **Ultima**, a **collective-communication** system for DDP All-Reduce that ensures **bounded**, **predictable completion times** in the presence of myriad computation and communication variabilities. Our approach **exploits** the stochastic **nature** of **machine learning** systems to work with **approximated gradients** (as previously exploited by gradient sparsification, gradient compression, hardware design optimization, and in-network aggregation techniques) to provide an efficient **balance between (tail)** performance and the **resulting accuracy** of the trained models.

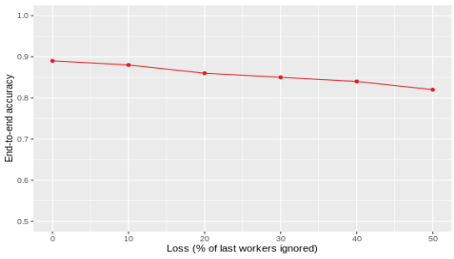
2. Existence of tail

- We test our hypothesis by looking at the All-Reduce **step-times** for our **proposed** vs **state-of-the-art collective** algorithms. We then **drop the tail** of a distributed deep learning job and witness its **effect on speed-up** and **accuracy**.
- The evaluations are performed in Nvidia's HPC environment, with 128 GPU nodes on a workload of ResNet50 with ImageNet.

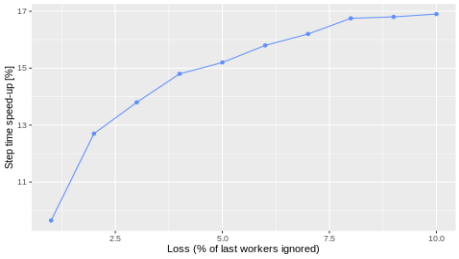
a. Tail in all-reduce algorithms



b. Tail-drop vs accuracy



c. Tail-drop vs speed-up

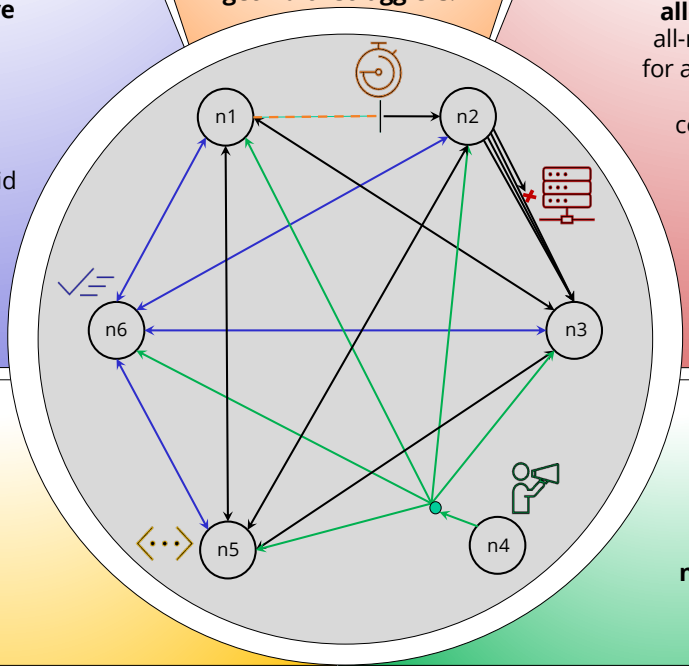


3. Our proposal: **Ultima**

1. Transpose All-Reduce
We proposes Transpose All-Reduce as the **collective algorithm** in our system which offers a **fixed O(1) hops per all-reduce call**. This not only helps in avoiding the tail, but also enables our system to avoid cumulative losses.

2. Dynamic Timeouts
Adaptive timeouts are used to **time-bound** every training step and **get rid of stragglers**.

3. Bounded Transport
Our design is motivated by **allowing some losses** in the all-reduce step, hence we opt for a lossy transport to **get rid of re-transmissions** and connections, **eliminating** a major cause of **tail** in the system.



5. Hadamard transform
We send out **encoded gradients** every all-reduce call to help **evenly spread any losses** occurred among the gradient layers.

4. Native Multicast
Using lossy transport, we are able to do **native multicast** in our system which **reduces** the **number of copies** needed to be sent out in the network.

Email: ewarraic@purdue.edu