Parameter Hub: High Performance Parameter Servers for Efficient Distributed Deep Neural Network Training

Liang Luo*, Jacob Nelson†, Luis Ceze*, Amar Phanishayee†, Arvind Krishnamurthy*

*University of Washington, †Microsoft Research

ABSTRACT

Most work in the deep learning systems community has focused on faster inference, but arriving at a trained model requires lengthy experiments. Accelerating training lets developers iterate faster and come up with better models.

DNN training is often seen as a compute-bound problem, best done in a single large compute node with many GPUs. As DNNs get bigger, training requires going distributed. Distributed deep neural network (DDNN) training constitutes an important workload on the cloud. Larger DNN models and faster compute engines shift the training performance bottleneck from computation to communication. Our experiments show existing DNN training frameworks do not scale in a typical cloud environment due to insufficient bandwidth and inefficient parameter server software stacks.

We propose PHub, a high performance parameter server (PS) software design that provides an optimized network stack and a streamlined gradient processing pipeline to benefit common PS setups, and PBox, a balanced, scalable central PS hardware that fully utilizes PHub capabilities. We show that in a typical cloud environment, PHub can achieve up to 3.8x speedup over state-of-the-art designs when training ImageNet. We discuss future directions of integrating PHub with programmable switches for in-network aggregation during training, leveraging the datacenter network topology to reduce bandwidth usage and localize data movement.

1 DISTRIBUTED DNN TRAINING IS COMMUNICATION BOUND

The goal of this work is to accelerate distributed DNN training in cloud environments. This work focuses on “data” parallelism, where workers process different samples and share the same model. A training iteration in this paradigm has two main components: computation-heavy forward and backward passes, and a communication-heavy model update step. As DNN models get larger and speedier accelerators emerge, the performance bottleneck of distributed DNN training has shifted from computation to communication.

Table 1: Major DNN training frameworks have similar throughput for training ResNet-50 with SGD (in samples per second, using a 56Gbps IP-over-InfiniBand network and one GTX 1080 Ti GPU per worker).

<table>
<thead>
<tr>
<th>Framework</th>
<th>Local</th>
<th>2 workers</th>
<th>4 workers</th>
<th>8 workers</th>
</tr>
</thead>
<tbody>
<tr>
<td>TensorFlow</td>
<td>152</td>
<td>213</td>
<td>410</td>
<td>634</td>
</tr>
<tr>
<td>Caffe2</td>
<td>195</td>
<td>266</td>
<td>343</td>
<td>513</td>
</tr>
<tr>
<td>MXNet</td>
<td>190</td>
<td>187</td>
<td>375</td>
<td>688</td>
</tr>
</tbody>
</table>

Larger DNN models require more gradient communication per iteration. The throughput of GPUs on ResNet, a recent DNN, has increased by 35x on modern cloud-based GPUs (Figure 1a), effectively demanding a similar increase in network bandwidth given a fixed batch size. However, the network bandwidth in compute instances on major cloud providers such as EC2 has not improved across generational upgrades [2]. Further, existing parameter exchange mechanisms have problems scaling up the total throughput on a standard cloud network stack (Table 1). The compound effect of these factors dramatically increases communication overhead during DDNN training.

Figure 1b summarizes the throughput of modest-scale DNN training with 8 workers and 8 colocated PSs on EC2 with 10Gbps links and a per GPU batch size of 4 (maximizing GPU memory usage on GRID 520): although modern DNN training frameworks can overlap backward passes with model updates, they can no longer hide the latency of communication due to faster computation. One solution is to increase the per GPU batch size, leading to a larger global batch size given a fixed number of GPUs. Large global batch sizes hurt statistical efficiency [3, 6, 7]; also, GPUs have limited memory. Techniques such as [5] could alleviate that pressure, but at a higher computational cost.

Communication overhead will likely worsen as the gap between computation and communication capability widens. New accelerators continue to reduce computation time, but networks are not getting faster at the same rate. Over the last 5 years, 100 Gbps networks have become available, but they pose high cost and have limited deployment.

These observations suggest that DDNN training has shifted from a compute-bound problem to one that also has a significant network-bound component. It is critical to perform model updates efficiently.

2 PHUB: OPTIMIZED PARAMETER SERVER

Model updates are usually performed in a parameter server (PS), a key-value store for the current model [10, 11, 15, 16]. We base our work on MXNet, a widely used, state of the art DDNN training framework that is known to be fast (Table 1, [4, 14]) and supports
native distributed training. Our profiling of MXNet reveals two problems: (1) insufficient network bandwidth (more so with colocated PSs\(^1\) than non-colocated servers) and (2) an inefficient PS software stack. We found that data copy, aggregation, and optimization are the main bottlenecks in the model update process: they prevent the PS from scaling to higher throughput with high bandwidth networks.

We propose PHub, a high performance PS design for DDNN training. We briefly summarize the main optimizations in PHub.

**Network Stack** Optimized InfiniBand support for lower network overhead, with one shot registration, zero copy, and minimized metadata, so all bandwidth is dedicated to gradient payload.

**Aggregation and Optimization** Fine grained key chunking (32KB) for maximized overlap of gradient processing and network transfer, and optimal load balancing in processor cores; locality-preserving, vectorized implementation of aggregator and optimizer.

**Gradient Memory Layout** NUMA aware, balanced scheme for assigning a key chunk to a processor core, through a series of load-balanced, locality-preserving assignment of queue pairs, interfaces, completion queues to cores (Figure 2a). PHub incurs zero synchronization between cores or between NUMA domains.

These software optimizations benefit centralized or sharded PS\(^2\) configurations. However, to scale up a central PS, software alone is not sufficient: the hardware in a typical server is unbalanced, with significantly more computing resources than network resources. Typically, a single interface/connection in the server must handle traffic for all participating workers. We propose PBox, a new server architecture that balances IO, memory and network bandwidth. Our prototype PBox is built using a server with ten 56Gbps network interfaces—5 per NUMA domain (Figure 2b). PBox takes full advantage of PHub software and essentially forms micro-shards inside a

---

\(^1\)The PS process and the worker training process reside in the same machine.

\(^2\)Multiple PS processes, each in charge of a partition of keys.
REFERENCES


