PLink: Discovering and Exploiting Datacenter Network Locality for Efficient Cloud-based Distributed Training

Liang Luo[1] **Peter West**[1] **Arvind Krishnamurthy**[1] **Luis Ceze**[1 2] **Jacob Nelson**[3]

Abstract

Training deep learning models has become an important workload on the public cloud. Scaling cloud-based distributed training faces unique challenges from the hierarchical network topology of the datacenter and the dynamic nature of the multi-tenant environment. Timely training of deep learning models requires effective use of topology-induced locality in the datacenter network. This work proposes PLink, an optimized communication library that probes the physical network then generates and executes a fitted hierarchical aggregation plan to take advantage of such locality, and evolves the plan to adapt to changing network conditions. PLink needs no support from cloud providers and operates out-of-the-box on unmodified public clouds. PLink serves as a direct plug-in to many training frameworks, delivering up to 2.3x better end-to-end training throughput for popular deep learning models on Azure and EC2 compared to state of the art.

1 Introduction

The distributed training pipeline consists of two stages: local computation (e.g., forward and backward passes) and global communication (parameter exchange). Efficient distributed training involves optimizing both stages. Past work has focused on building specialized hardware clusters with fast interconnects (Smith et al., 2017; You et al., 2018; Akiba et al., 2017; Jia et al., 2018a; Mikami et al., 2018; Sun et al., 2019; Goyal et al., 2017; Iandola et al., 2016). While the results are encouraging, these approaches demand steep investments and are not universally available.

On the other hand, public cloud-based learning has become a popular, more accessible alternative, as all major cloud providers offer racks of nodes with specialized accelerators (such as GPUs, TPUs, and custom FPGAs) (Google; Microsoft, c; Amazon, a;e; Fowers et al., 2018; Jouppi et al., 2017; Xiao et al., 2018a) for deep learning workloads.

Unfortunately, even with modern frameworks and recent optimizations (e.g., gradient compression and quantization (Lin et al., 2017; Seide et al., 2014; Lim et al., 2018), latency-hiding (Zhang et al., 2017; Jayarajan et al., 2019; Hashemi et al., 2018), optimized communication libraries (Facebook; Nvidia; PSLite) and large batch optimizations (Goyal et al., 2017)), distributed training at scale on the public cloud still incurs high overhead: up to 90% of total training time can be wasted waiting on the network (Figure 1). While existing solutions focus solely on addressing the *bandwidth* bottleneck, the cloud environments pose additional challenges (§3). The hierarchical network structure breaks the usual assumption of link speed being uniform; multi-tenancy and the dynamic nature of the cloud traffic cause high variation in performance. All these add to the complexity of scaling up distributed training and can render existing solutions less effective (§3.1). These issues, coupled with the fact that the speed at which the throughput of emerging accelerators is improving significantly outpaces that of the networking devices (Luo et al., 2017), keep cloud-based training communication bound.

Our work focuses on designing a communication scheme for efficient gradient aggregation in the context of public clouds. We address three specific challenges in developing our communication scheme. First, we need an aggregation mechanism that is appropriate for network topologies that display bandwidth oversubscription, and that makes appropriate use of underprovisioned links. Second, we need to be able to identify the underlying network topologies and bandwidth/latency constraints (or collectively, locality) even if the public cloud does not expose such information. Finally, we need communication schemes that can react to changing network conditions, especially in the presence of interfering traffic generated by other tenants.

We propose PLink, an optimized, locality-aware, dynamic system that uses a hierarchical aggregation scheme for cloud-based, high-performance deep learning (DL) training. PLink uses three components to address the challenges above: (1) ProbeEmbed (§4.1), a general-purpose technique that uses

[1]University of Washington [2]OctoML [3]Microsoft Research. Correspondence to: Liang Luo <liangluo@cs.washington.edu>.

Proceedings of the 3^{rd} MLSys Conference, Austin, TX, USA, 2020.

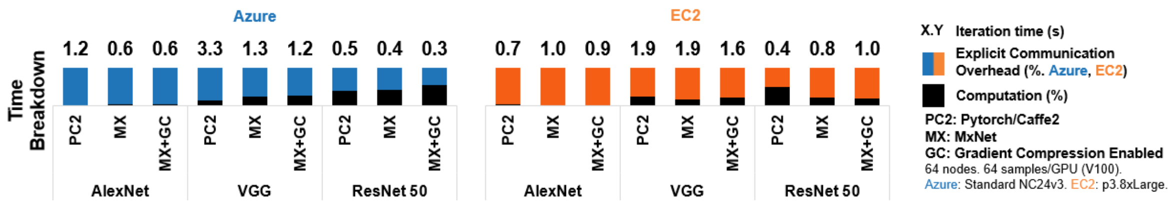

Figure 1. Even with state of the art training frameworks and recent optimizations, up to 90% of time during cloud-based training of popular models is wasted explicitly waiting on the network.

data from measurement probes to embed cloud VMs in a multi-dimensional space in order to accurately infer network topology and group VM nodes based on their physical affinity; (2) AggEngine (§4.2), a hierarchical aggregation execution engine codesigned with the DL training workload that generates and executes a balanced hierarchical aggregation plan using topology information; (3) Autotune (§4.3), a reactive mechanism to quickly adjust to network changes by rebalancing the aggregation workload between VMs and continuously fine-tuning aggregation communication paths to match each VM's current network performance.

PLink serves as a direct plug-in to popular frameworks and requires no support from cloud providers. On unmodified public clouds (both Azure and EC2), PLink is able to deliver up to 95% accuracy in inferring datacenter network topology, and 2.3x faster end-to-end training performance for popular DL models compared to state of the art.

2 Background

We provide an overview of typical structures and performance implications for datacenter networks and common approaches to the communication phase of training.

2.1 Datacenter Networks

A typical datacenter network has a hierarchical, multi-tiered topology (Mysore et al., 2009; Greenberg et al., 2009; Roy et al., 2015; Liu et al., 2017). Machines are grouped into *racks*, each connecting to a top-of-rack *(ToR)* switch. ToR switches are connected to upper level devices. In this setting, the communication performance of two end-hosts is affected by where they reside: they enjoy full link bisection bandwidth within a rack, because link capacity is not shared at the rack-level, but if they are on different racks, the performance depends on link load and oversubscription ratio (Bilal et al., 2012). In this paper, we use *locality* to refer to the cause of variation in communication performance, which includes: (1) physical topology: the location of the nodes and (2) dynamic network load. Efficient communication requires carefully architecting software to tap into both aspects (Jeyakumar et al., 2012; mixpanel): solutions that ignore physical topology are subject to long-term communication imbalances, and those that ignore dynamic network load suffer from short-term inefficiencies.

2.2 Parameter Exchange in Distributed Training

Parameter exchange is the bottleneck phase in cloud-based distributed training. The underlying mechanism for parameter exchange can be classified into one of the following:

Parameter Servers (PS) (Smola & Narayanamurthy, 2010; Li et al., 2014a;b; Zhang & Ré, 2014; Luo et al., 2018; Zhang et al., 2017; Cui et al., 2016). PSs are centralized or sharded key-value stores, where keys and values represent the model's layer IDs and weights. In each iteration, all workers update the model stored in PSs with their gradients.

Collective AllReduce (CA) (Sack, 2011; Thakur et al., 2005; Rabenseifner, 2004; Blum et al., 2000; Bala et al., 1995). Popular in the context of MPI, all nodes in CA participate in the communication, usually running symmetric tasks. The end goal of CA is that all nodes have a globally-reduced copy of the data. Widely used CA in training deep learning models include halving-doubling (Goyal et al., 2017), ring, and double binary tree (Nvidia; Sergeev & Balso, 2018).

Hierarchical Aggregation (HA). Pervasive in the HPC world (Graham et al., 2016), HA refers to the generic technique of aggregating data in multiple steps, first locally and then globally. Examples of HA usage for distributed training include Iandola et al. (2016); Cho et al.; Geng et al. (2018); Wangt et al. (2018), though not in a cloud datacenter context, and all require the network topology to be known.

The existing approach to using a given communication paradigm in the cloud is straightforward. The user requests a list of VMs from the cloud provider directly or from a service (e.g., Batch AI (Microsoft, a), Dynamic Training (Amazon, c)), and then the list of node addresses are used to launch a deep learning framework. Therefore, this unordered list determines the identity and rank of each node, which dictates the communication pattern.

3 Motivation

Two major challenges exist in cloud-based training.

Non-uniform link bandwidth. Host-to-host bandwidth in the cloud is non-uniform due to the hierarchical structure of

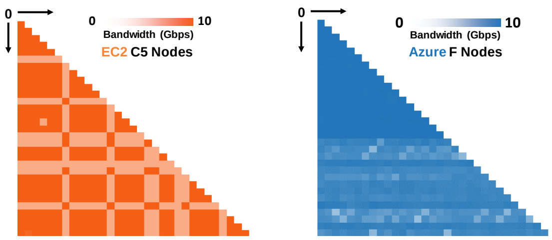

Figure 2. Pairwise bandwidth probes with 32 EC2 C5 and Azure F instances show non-uniform link bandwidth.

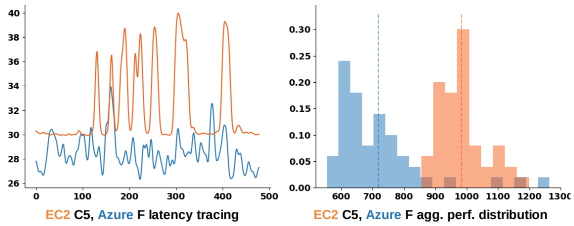

Figure 3. Left: 8 hour latency (us) tracing (1 minute average) between two VMs on both clouds show steep fluctuations due to volatile cloud traffic. Right: Wide performance distribution of the same, periodically launched Gloo aggregation task on both clouds.

the datacenter (§2.1). Figure 2 shows a pairwise bandwidth probe of 32 VMs in EC2 and Azure, in the same availability zone/datacenter. In both cases, faster pairs can deliver more than 2x the throughput of slower pairs.

Volatile traffic. The performance variability in the public cloud is well known (Farley et al., 2012; Iosup et al., 2011; Maricq et al., 2018). Although mechanisms for performance isolation have been proposed (Shieh et al., 2010; 2011), we still observe interference from other workloads, leading to volatile latency and aggregation performance (Figure 3).

3.1 Inefficiencies in Existing Approaches

We motivate our design by analyzing why some existing approaches do not perform optimally, as they rely on assumptions that aren't typically valid in the datacenter setting.

Figure 4 shows a theoretical analysis of widely used communication patterns. PS (a) and popular choices of CA such as halving-doubling (b) and ring (c) are shown in a setup where nodes (0-3, enclosed in a circle) are spread equally among two clusters (purple and gold). The left side of the figure shows patterns that achieve optimal locality in the setting by exchanging data among nodes with high locality (high-performance links in green) while minimizing transfers over the bottleneck links (slow links in red). The right side shows alternative reduction routes with poor locality. All patterns achieve the same result, but with different efficiency.

The problem with poor locality happens when the communication pattern in the algorithm is not optimally aligned with physical topology. Mapping logical ranks to physical hosts

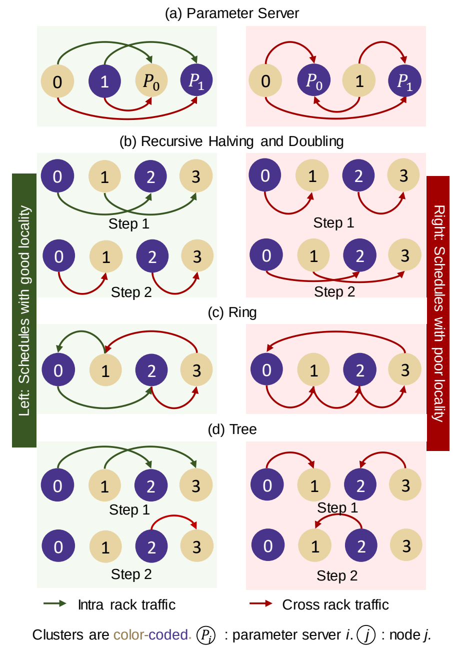

Figure 4. Existing aggregation approaches can suffer from poor locality if not taking physical network topology into account.

in a locality-preserving way is contingent on awareness of the physical network structure. Hence, *topology-awareness* is crucial for efficient aggregation in a datacenter network.

Even with careful mapping, not all algorithms work optimally in the datacenter environment. Table 1 summarizes network characteristics of these algorithms, with a simplified, flattened datacenter network topology model where nodes are simply placed in different racks. Centralized PSs are known to suffer from *incast* congestion and do not scale to a large number of workers (Iandola et al., 2016; Geng et al., 2018; Luo et al., 2018). Sharded PSs incur high cross rack traffic. CAs usually trade off lower per-link traffic on the wire with more rounds of communication, which is not suitable when the latency is high. Tree reduction inherits the problems of both PS and CA: high fan-out causes incast problems; a low fan-out adds more rounds. Ultimately, we need an algorithm that bounds communication steps, takes advantage of fast links, and localizes traffic to avoid interference from competing traffic.

3.2 2-level Hierarchical Aggregation (2LHA)

HA is not new, but most applications of 2LHA are in contexts with known network topologies. HA does not reduce the total amount of data transferred on the wire, but it can create more *localized* traffic and avoid slow links.

One important parameter in HA is the number of levels. Similar to (Alfatafta et al., 2018), we choose 2 as the level based on our domain knowledge of datacenter networks, that oversubscription mostly hurts at the rack level. Thus,

Name	Rounds/Hops	Bytes on Wire	XR Bytes
PS (fully sharded)	2	$2(NC-1)S$	$2N(C-1)S$
Halving doubling	$2log_2 NC$	$2(NC-1)S$	$2(C-1)S$
Ring-Chunked	$2NC-1$	$(2NC-1)S$	$(2C-1)S$
Tree (fan out = C)	$2(log_C N+1)$	$2C\frac{NC-1}{C-1}S$	$\approx 2C\frac{(N-1)}{C-1}S$
2-level hierarchical	4	$2S(NC-1)$	$2(C-1)S$

Table 1. Network characteristics of various algorithms featuring rounds of communication (Rounds), minimum total traffic (Bytes on Wire), and minimum cross rack traffic (Min XR Bytes, corresponding to red arrows in Figure 4) to allreduce with NC nodes on C racks, each with N nodes. Each node has a buffer of size S bytes. PS and aggregators in HA are colocated and sharded.

by separating inter- and intra-rack aggregation, we can best capture the static aspect of locality and minimize latency. The use of more levels (> 2) suffers from higher latency and volatile performance, as messages need to traverse multiple links with unpredictable latency, but provides no benefit compared to PS if links don't have enough non-uniformity (e.g., in the same cluster).

2LHA partitions nodes into different groups (clusters) based on their affinity. 2LHA starts by chunking the buffer across members in the same group. For each chunk, a node is designated as the local master (LM) for that group, for aggregating locally. One of the LMs across all groups is chosen as the global master (GM) for global aggregation. Visually, the reduction trees of all chunks form a 2-level forest. Communication for 2LHA is done in the following steps:

1. Each group member sends all chunks to their respective LMs (intra-group traffic only).
2. All LMs send per-group aggregated chunk to the GM for global aggregation (inter-group traffic only).
3. The GM aggregates the chunk, then uses the reversed routes for propagating the globally-aggregated chunk back to the LMs.
4. The LMs fan out the globally aggregated chunk to all group members.

Efficient execution of 2LHA requires overlapped intra- and inter-group aggregation and load-balanced LM and GM assignments. §4 provides an implementation that satisfies these. Table 1 shows the desirable properties of 2LHA. Compared to CAs, the number of rounds in 2LHA is constant with respect to the number of nodes, compared to PSs, it requires significantly less cross-rack bandwidth.

4 Design and Implementation

We now describe PLink, a topology-aware and dynamic system that leverages HA for efficient cloud-based training. PLink includes three components.

- ProbeEmbed: a network probing and clustering approach to capture physical locality in the datacenter network. ProbeEmbed groups nodes based on their physical affinity, so intra-group links have better communication performance than inter-group links.
- AggEngine: a high-performance implementation of 2LHA that is codesigned to take advantage of deep learning properties. AggEngine uses clustering information to distribute the aggregation workload efficiently and execute the aggregation schedule.
- Autotune: a mechanism that tracks training performance and adjusts the current GM and LM assignments to adapt to changes in the network conditions.

4.1 Capturing Network Locality with ProbeEmbed

For accurate network topology discovery, ProbeEmbed must probe quickly and should not rely on knowledge of a particular datacenter. ProbeEmbed: (1) probes communication links between nodes to measure pairwise node distances, (2) denoises probed distances, and (3) clusters nodes.

4.1.1 Running ProbeEmbed probes

ProbeEmbed starts by issuing measurements to identify communication locality and determine pairwise node distances. Distance is defined using universal networking concepts, like latency or inverse bandwidth. ProbeEmbed uses two different probes: an inhouse DPDK-based (dpdk) probe to provide near bare-metal latency measurements for supported VMs on Azure and EC2, and iPerf (iperf). ProbeEmbed runs these networking probes one-to-one. $O(N^2)$ time would be required to probe N nodes if run sequentially. To accelerate this process, ProbeEmbed picks as many pairs (up to $\frac{N}{2}$) as possible in each round without having a node appear twice, to avoid interference from concurrent tests. This allows ProbeEmbed to probe in $O(N)$ rounds.

ProbeEmbed derives pairwise distances with probe results (in case of bandwidth measurements, bandwidth are converted to distance by taking the inverse (scikit learn)). ProbeEmbed then proceeds to *denoise* the collected data.

4.1.2 Denoising probe data with embedding

ProbeEmbed embeds nodes in a Euclidean coordinate space, obtaining a set of coordinates whose distances agree with the probed distances. This works to: (1) denoise measurements by leveraging Euclidean space to approximate the physical location of nodes; and (2) obtain a set of "virtual coordinates" for a clustering algorithm to identify groups.

To embed nodes, we identify node coordinates ($\mathbf{v_i}$ and optional h_i) that minimize the following objective:

$$\sum_{i=1}^{n}\sum_{j=1}^{i-1}\left((d_{i,j})^{\alpha} + \mathbb{1}_h\left[h_i + h_j\right] - p_{i,j}\right)^2 \qquad (1)$$

where n is the number of nodes, $d_{i,j} = \|\mathbf{v_i} - \mathbf{v_j}\|_2$ is the

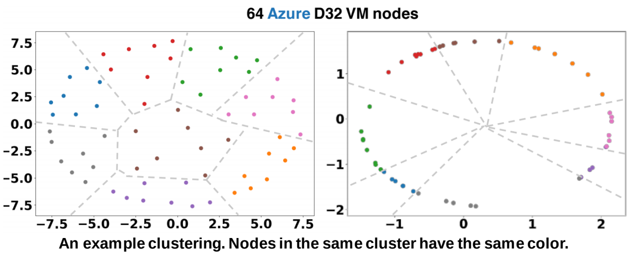

Figure 5. $\alpha = 2$ (right) generates more consistent clusters (higher AMI across 100 generated clusters) compared to $\alpha = 1$ (left)

Euclidean distance between embedded node coordinates for nodes i and j, $p_{i,j}$ is their probed distance, parameter α takes a value between 1 and 2, h_i is a non-negative startup cost parameter for node i, and $\mathbb{1}_h$ is a switch for h_i. We use the Adam algorithm (Kingma & Ba, 2014) to optimize this.

ProbeEmbed embeds VMs in a coordinate space that preserves the probed distances between VMs. The denoising effect of the embedding process stems from its tendency to keep mutually close nodes together, which enforces our domain knowledge that VMs that are close to one particular reference VM are probably close to each other as well in the datacenter. Thus, this effect has a correcting influence when the mutual-closeness property is violated by a particular observation but is observed in a majority of nodes. A lower number of embedding dimensions strengthens this effect.

α tunes how longer distances are treated in the Euclidean space: $\alpha = 1$ fits the embedded distances to probe distances exactly. For $\alpha > 1$, we can achieve increased "compaction" of distance while maintaining relative distance order. This effect is desired because small physical distances can be magnified disproportionately in the probe due to competing traffic. Setting $\alpha = 2$ causes the long probed values to be "compacted" more than the smaller ones (but it never changes the relative order of distances). Figure 5 suggests empirically, a larger α pushes VMs with short distances even closer on the embedded plane, leading to more consistent clusters evidenced by higher adjusted mutual information (Vinh et al., 2010) (0.59 vs 0.76) across 100 runs.

Inspired by (Dabek et al., 2004), ProbeEmbed includes an optional parameter h_i, the node-specific, network-agnostic, fixed latency of sending a packet (e.g. traversing the operating system network stack).

Multiple clusters are generated at once, and ProbeEmbed favors clusters with smaller diversity in the sizes of groups. If there is a tie, ProbeEmbed makes a random choice.

4.1.3 Grouping Nodes for 2LHA

We now outline how ProbeEmbed partitions VMs into groups for 2LHA. The goal is to generate groups that are (1) balanced, so no single group becomes a bottleneck and (2) cohesive, so VMs in the same group have good locality. We first compute the number of groups to generate, then determine the members of each group.

GMs are more likely to be the bottleneck during 2LHA, as they must receive and aggregate messages at both levels. For a uniform key distribution, the following term approximates the bytes-on-wire sent or received by a GM:

$$b = O\left(\frac{n}{k} + k\right) \tag{2}$$

with n total VMs, and k groups. The GM sends and receives a message from each node within its group ($\frac{n}{k} - 1$) for local aggregation, and from every other group ($k - 1$) for global aggregation. This expression achieves a minimum at $k = \sqrt{n}$, giving a natural choice for group count.

Once group count is selected, we use a constrained k-means clustering algorithm with k-means++ initialization (Bradley et al., 2000; Arthur & Vassilvitskii, 2007) to generate balanced, locality-preserving groups. This accepts a minimum cluster size and the number of groups to generate as input. For perfect balance, both parameters are set to $\sqrt{n}$.

Enforcing perfect balance is not optimal in cases where VMs are naturally clustered in almost balanced but distant clusters, because in those cases some group can contain a distant member which could be assigned to a much more cohesive group with a slight imbalance, forcing an onerous bottleneck on the local aggregation step. Thus, we include a parameter, *balance elasticity*, which enables a slight imbalance among clusters. We empirically found best results with values between 1.0 (perfectly balanced) and 2.0 (each group has at least $\sqrt{N}/2$ nodes).

In order to evaluate the performance of ProbeEmbed, we also define Balanced Random, a grouping method for 2LHA that operates without considering probe distances and simply produces $\sqrt{n}$ groups of size $\sqrt{n}$ uniformly at random. This is used later as a baseline.

4.2 Efficient HA with AggEngine

AggEngine transforms grouping information from ProbeEmbed into a hierarchical reduction plan and efficiently executes it. AggEngine supports various communication backends, including TCP, RoCE (SoftRoCE), iWarp (Metzler et al., 2010), and InfiniBand.

4.2.1 Generating an Aggregation Plan

AggEngine chunks buffers for better load-balancing across processor cores and overlapping of the transmission of gradients with aggregation.

Since AggEngine is bottlenecked by the slowest inter-group transfer, which in turn is bottlenecked by the slowest intra-group transfer, AggEngine assigns chunk GMs and LMs such that each group (or node) has a number of GMs (or LMs) proportional to its cardinality. AggEngine uses an approximation set partition algorithm to achieve this.

AggEngine then generates a schedule that executes the steps in §3.2 for each chunk. A schedule consists of a set of chunk-action pairs, where action is one of the following[1]:

- **SendTo(nids)**: send the content in the current *merge buffer* to the list of nodes specified in *nids*. SendTo is a non-blocking operation, and its status is inferred by whether subsequently anticipated data is received.
- **ReceiveFrom(nids)**: block until the chunks from *nids* are received and aggregated into the merge buffer.
- **Fetch**: notifies AggEngine that a framework-supplied buffer is ready to be processed.
- **Deliver**: writes the content in the merge buffer back to the framework-supplied buffer.

A schedule is represented as a DAG where dependencies are edges and nodes are primitives, avoiding false dependencies between local and global aggregation.

4.2.2 Executing an Aggregation Schedule

AggEngine first performs rendezvous with an out-of-band mechanism (e.g., Redis), establishing multiple connections per pair of VM as cloud providers can restrict per-stream bandwidth (Amazon, d). AggEngine preserves intra-node locality by maintaining a load-balanced map from a chunk to a connection, and then by further associating the connection to a particular processor core (Pesterev et al., 2012).

We now focus on how AggEngine efficiently supports the four actions, hiding communication latency and avoiding excessive synchronization.

When a framework calls `reduce(chunk)`, AggEngine retrieves the thread ID, suspends it, and enqueues `chunk` to the ready queue. Its worker threads poll the ready queue to retrieve `chunk` and transition the buffer into the Fetch state, copying gradients from the supplied address, then set the state of buffer to SendTo.

SendTo is an asynchronous operation that simply enqueues data to a send FIFO queue. A cursor is used for each TCP connection if a send operation cannot finish. AggEngine enforces that the order of bytes on the wire corresponds exactly to the order in which SendTos are issued. This saves metadata overhead as only a 4B integer per flow that encodes the chunk ID is required.

SendTo is followed by ReceiveFrom. AggEngine allows streaming aggregation for each buffer in ReceiveFrom state to a *merge buffer*. A counter is incremented when a chunk is fully received from a peer, and when the counter reaches the target, this step concludes. AggEngine transitions current buffer state to SendTo or Deliver based on schedule.

[1]With these action primitives, AggEngine can support arbitrary reduction graphs.

Property	AggEngine Optimization
Fixed comm. pattern	No explicit acknowledgement
Fixed buffer size	Minimal metadata
One reduction per layer per iteration	Only 2 merge buffers per layer; Eagerly accepts chunks
Training is stochastic	Switching plans quickly and cheaply

Table 2. Codesigning AggEngine with training workload.

The last step of a schedule is Deliver, where AggEngine copies the final value to the framework-supplied address. AggEngine then wakes up the thread that called `reduce`. AggEngine alternates between two merge buffers per chunk for synchronous training to overlap local computation on a chunk with transfers of that chunk to peer nodes.

Table 2 summarizes how AggEngine is designed to take advantage of properties in the distributed training workload to lower its overhead.

4.3 Reacting to Network Changes with Autotune

Autotune collects performance information from AggEngine and watches for sudden changes in link conditions, reflected by the current training speed. The goal of Autotune is to dynamically compensate for link changes by redistributing LMs and GMs to VMs, so the time to finish an iteration is similar at both local and global levels.

A perfect initial LM and GM assignment is hard, even if we have bandwidth probe measurements. Consider an aggregation plan S, where the *effective bandwidth* of node i to j while running aggregation S is $BW(i,j)$. Clearly, finding the best S analytically relies on $BW(i,j)$ to be precisely measured or modeled, but $BW(i,j)$ has a circular dependency on S itself. Autotune thus makes approximations when optimizing the assignments.

At a high level, Autotune works in two phases: (1) a *Quicktune* phase where a one-shot, global adjustment of GM and LM assignments is done to adapt to the network change immediately; and (2) a *Finetune* phase where Autotune uses a performance model to find the current performance bottleneck in the system, and moves GMs and LMs away from it in a stepwise, increasingly aggressive manner.

4.3.1 Quicktune

Quicktune aims to minimize the maximum transfer time of each node. Quicktune can be best summarized formally as follows: let $GM(i,c) \in \{0,1\}$ and $LM(i,c) \in \{0,1\}$ be the boolean variables to be solved, which indicate whether node i is the GM or LM of chunk c. Let $G(i)$ be the group of node i, $|G|$ the number of groups and $S(c)$ the size of c

in bytes. Our goal is to:

minimize

$$max(t_i = \frac{\sum_c S(c)(LM(i,c)|G(i)| + GM(i,c)|G|)}{\sum_n BW(i,n)})$$

subject to

$$\forall_{i,c} \quad GM(i,c) = 1 \implies LM(i,c) = 1$$

$$\forall_c \quad \sum_i GM(i,c) = 1$$

$$\forall_{c,g\in G} \quad \sum_{i\in G(g)} LM(i,c) = 1$$

Quicktune solves this with an approximation: it first distributes GMs to different groups, with the number of GMs assigned to each group proportional to the group aggregate bandwidth $\sum_{m\in G(i)} \sum_{n\notin G(g)} BW(m,n)$, then distributes LMs inside each group to different members in a similar fashion, using aggregate per node bandwidth.

4.3.2 *Finetune*

Quicktune is limited as it assumes constant effective bandwidth across different schedules and ignores node balance. Finetune, however, amends this by gradually *evolving the current schedule*, using both the currently measured $D(i,j)$ and $B(i,j)$ to pinpoint the current bottleneck node in the system, and then moves away its load while maintaining balance based on *blame*. Blame for node i ($B(i)$) has two major weighted parts: *time* $t(i)$ and *imbalance* $l(i)$. Autotune collects per connection stats including link $RTT(i,j)$, bandwidth $BW(i,j)$ through the OS (Wikipedia; Mathis et al., 2003), and computes $t(i) = max_j RTT(i,j) + \frac{\sum_j D(i,j)}{\sum_j BW(i,j)}$ and normalized $\hat{t(i)} = \frac{t(i)}{mean_n t(n)}$. Further, we let $l(i) = \sum_j D(i,j)$ and weighted $\hat{l(i)} = \frac{l(i) - min_n l(n)}{max_n l(n) - min_n l(n)}$. Finally, blame is defined as $\beta \hat{l(i)} + \gamma \hat{t(i)}$.

With blame for each node available, Finetune attempts a move of a single GM (or if not available, an LM) from the node with highest blame to the lowest, if $\frac{max_i B(i)}{min_i B(i)} > \epsilon$ (a configuration parameter). If Finetune repeatedly identifies the same bottleneck node, it moves exponentially more LMs and GMs in each step.

Autotune uses a central daemon to collect performance metrics and generate new schedules, and is triggered by a sudden change (e.g., larger than 20%) in training performance. Autotune signals AggEngine to install the new schedule.

4.4 Integration with Training Frameworks

Integrating PLink with a training framework is straightforward. PLink's initialization requires only a list of nodes and buffer sizes/addresses from the training framework. Here we demonstrate how PLink can be integrated with systems with training frameworks that have different design decisions.

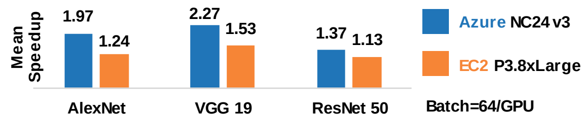

Figure 6. PLink's collective optimizations achieve up to 2.27x speedup on public clouds training popular neural network models, compared to original Pytorch/Caffe2.

Caffe2/Pytorch uses blocking reduction. PLink integration is done by extending Gloo (Facebook)'s `algorithm` object. Caffe2/Pytorch, by default, uses a concurrency limit to control the number of established connections per peer and simultaneous `allreduce` calls. PLink instead uses a single AggEngine instance to multiplex all requests.

MxNet uses an asynchronous callback pattern, but PLink's `reduce` is blocking. A separate thread is launched for invoking callbacks through `kv_apps`'s infrastructure when a chunk has finished. Support for MxNet is enabled by extending PSLite (PSLite)'s `van` object.

5 Evaluation

Our evaluation goals are as follows: (1) Measure PLink's impact on end-to-end training. (2) Demonstrate the efficiency of AggEngine. (3) Quantify the benefit of hierarchical aggregation and topology-awareness. (4) Evaluate the accuracy of ProbeEmbed's inference of physical affinity. (5) Assess how well Autotune reacts to network changes.

5.1 Environment Setup

We run experiments on both Azure and EC2, in the same region or availability zone. Each VM runs Linux kernel 4.18 with SoftiWarp support (IBM), Cuda 9.2, CuDNN 7, and network enhancements (Microsoft, b; Amazon, b) if possible. All VMs have at least 10 Gbps network throughput, using DCTCP (Alizadeh et al., 2010) as SoftRoCE and SoftiWarp do not yield better performance. AggEngine uses a chunk size tuned to its best performance, usually 16 to 64KB. All experiments use 64 nodes unless specified. Some experiments involve comparing results generated with uncertainty. In those cases, we report speedup in terms of mean and percentile metrics together with performance distribution of 50 runs. Each sample represents mean performance of 20 iterations using a potentially different cluster assignment generated by the underlying mechanism.

5.2 End to end training performance

We start our evaluation with the impact of PLink's collective optimizations on the end-to-end training performance of

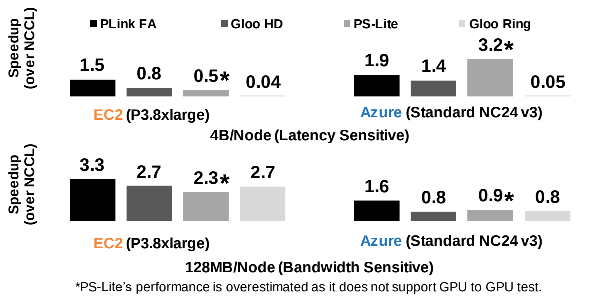

Figure 7. GPU to GPU aggregation speedup of various systems on Azure and EC2 in terms of mean latency for large and small buffers, normalized to NCCL's performance.

popular neural networks. Then we breakdown the effect of each optimization in the following sections. We compare PLink to original Pytorch/Caffe2 performance by replacing Gloo. We represent training speedup as mean speedup in throughput of 50 iterations to save experimental cost. This directly translates to a reduction in end-to-end training time as PLink's optimization does not change convergence because it keeps the computation intact.

Figure 6 shows the speedup of PLink, which ranges from 1.37-2.27x on Azure, and 1.13x-1.53x on EC2[2] when training popular vision models. We expect larger speedups for neural networks with higher communication to computation ratios (such as AlexNet and VGG) on VMs with faster networks, as we observe near line-rate network utilization when training them with PLink. For models with smaller communication to computation ratios (such as ResNet-50), PLink's speedup comes from its reduced reduction latency.

5.3 Efficiency of AggEngine

This section evaluates the performance of AggEngine itself, disabling ProbeEmbed and Autotune. For clarity, we first set up our baseline with communication libraries used in major training frameworks, including MxNet PS-Lite, Nvidia NCCL 2.4, and Facebook Gloo, covering ring (Gloo, NCCL), tree (NCCL), halving-doubling (Gloo) and parameter server (PS-Lite). We use each library's benchmark program to measure its performance. On both Azure and EC2, we use instances with V100 GPUs and test end-to-end reduction performance involving copying from/to a GPU.

Figure 7 shows the speedup in terms of mean (20 iterations) GPU to GPU reduction latency normalized to NCCL's result of aggregating a large (128MB) and a small (4B) buffer. To provide a fair comparison, we limit AggEngine to use a single connection per peer (in a typical scenario, multiple connections are used). Overall, AggEngine's flat aggrega-

[2]While we report only data-parallel results for its dominance in distributed training, PLink optimizations are paradigm-agnostic, as all the scheduling of transfers is done by the framework.

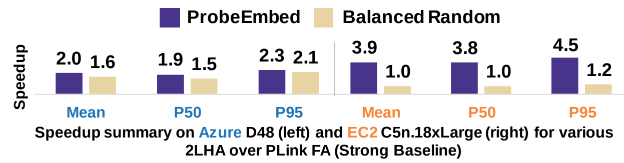

Figure 8. Mean, P50, and P95 additional speedup of 2LHA using different approaches over FA (strong baseline). Run on 64 Azure D64 series and EC2 C5n.18xLarge instances, with a constant VM topology throughout this experiment.

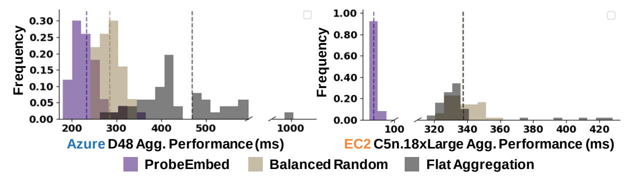

Figure 9. Empirical reduction latency distribution of ProbeEmbed-, Balanced Random- guided 2LHA and FA on Azure and EC2.

tion (FA) leads the pack and is thus used as a strong baseline.

5.4 Effectiveness of Hierarchical Aggregation

We continue with a detailed comparison between PLink FA and 2LHA, reducing a real-world model (ResNet-50, varying buffer sizes totalling ≈100MB). We use 4 connections per peer to fully utilize VM bandwidth. We separately tuned chunk sizes to ensure baseline perform well in both clouds. We present the speedup summary in Figure 8; then, each subsection details the benefits associated with the individual technique. Performance distribution is found in Figure 9.

Comparing with ground-truth-guided 2LHA. Cloud providers can expose topology information to assist in locality-aware scheduling and communication. We obtain ground-truth topology information from Azure. We look at the effectiveness of using physical topology on Azure to form a reduction schedule for 2LHA, grouping VMs based on their physical locality.

Our experiment shows that ProbeEmbed-guided 2LHA beats ground-truth-guided 2LHA by 1.15x on average. The performance of ground-truth-guided 2LHA relies on a "luck" factor related to the physical topology of the VMs: the more compactly the VMs are allocated, and the more balanced the allocation in each rack is, the better performance of ground-truth-guided 2LHA should be. But VM spawn location is in total control of the scheduler, and sometimes it is impossible to guarantee a compact allocation due to current VM occupancy. It is difficult to splice/split undersized/oversized racks for a more balanced 2LHA without probing for network properties, which ProbeEmbed does.

Comparing with Balanced Random-guided 2LHA. We observe an additional 1.3x and 3.9x expected performance gain of the clusters generated by ProbeEmbed over those of

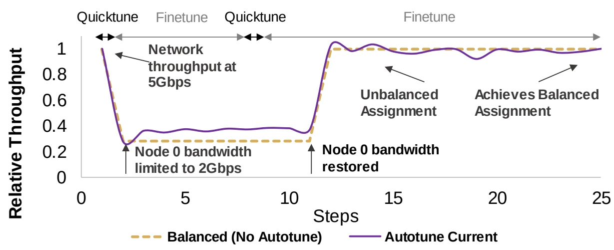

Figure 10. Autotune adapts to bandwidth changes by assigning LMs based on current metrics of AggEngine. Dotted lines: averaged throughput. Solid lines: snapshot throughput.

Balanced Random on Azure and EC2. ProbeEmbed leads to better and more stable performance by generating cohesive, locality-preserving group assignments. On the other hand, Balanced Random can include VMs that are far away in the same group, creating a bottleneck in the system.

5.5 Accuracy of ProbeEmbed

Effective exploitation of locality requires capturing both static and dynamic aspects of the network (which is directly captured by the probes). In this section, we evaluate ProbeEmbed's ability to discover the network topology in terms of *physical affinity*.

Physical Affinity Inference Accuracy (PAIA). We define *affinity score* as an intuitive metric to quatify how well ProbeEmbed captures network topology: for any two VMs (a, b) that have comparable distance to an observer c, let $T(a, c), T(b, c)$ be the ground-truth distance (in terms of hops) from a to c and b to c, and let $M(a, c), M(b, c)$ be the ProbeEmbed measured distance. We define an affinity score $A(a, b, c)$ for triplet (a, b, c) as:

$$A(a,b,c) = \begin{cases} 1 \textbf{ if } M(a,c) < M(b,c) \textbf{ and } T(a,c) < T(b,c) \\ \quad \textbf{or } M(a,c) > M(b,c) \textbf{ and } T(a,c) > T(b,c) \\ 0 \text{ otherwise} \end{cases}$$

For a given c, $A(a, b, c)$ captures whether ProbeEmbed's measurement agrees with the actual hop distance between a and b. $A(a, b, c)$ is only defined for *comparable* nodes a and b to c: each node's position is encoded as a tuple of format (region, datacenter, cluster, row, rack, host). a and b are comparable to c if they have different longest common prefix to c. Longer common prefix means higher affinity. We now define PAIA as the total sum of all $A(a, b, c)$ over the count of all defined $A(a, b, c)$s. A higher PAIA means a better comprehension of the datacenter network.

Quality of Various ProbeEmbed Probes. We evaluate base affinity accuracy achieved by running latency and bandwidth probes, without the embedding process, on 64 VMs, in 2 datacenters, 7 clusters, 15 rows and 61 racks in the US West 2 region with a total of 81.3K triplets to infer. Latency-based measurements better reflect underlying topology, with the DPDK Echo probe yielding a PAIA score of 95.6% compared to 77.4% with iperf, likely because bandwidth is not necessarily determined by distance.

Embedding's Effect on Affinity Inference. ProbeEmbed's probe readings can be affected by measurement noise. We now show how the embedding boosts ProbeEmbed's inference accuracy, in a case with 64 VMs in a single datacenter with 75.2K triplets. This effectively shrinks the latency distribution and makes it more difficult to infer topology. DPDK Echo probes without embedding yield a PAIA score of 81.3%. We apply embeddings with different α values to this dataset and found an average PAIA improvement of 5.0% with $\alpha = 1$, and 8.1% with $\alpha = 2$.

5.6 Effectiveness of Autotune

We conclude our evaluation with the real-world impact of Autotune on training ResNet-50 with Pytorch/Caffe2 on 8 nodes with FA. We report Autotune's effects in terms of end-to-end training performance. We run Autotune continuously, ignoring the stop condition, and assess how Autotune adapts to bandwidth changes and discovers a balanced LM assignment. We start by imposing no limits on bandwidth; then we limit the bandwidth of node 0, and eventually restore it. This shows how instantaneous training throughput changes as we apply Autotune decisions.

Figure 10 shows this process. We perform a step of Autotune every 10 iterations and report the snapshot reading of current throughput after the schedule change. At step=0, node 0's network throughput is ≈5 Gbps. At step=1, we limit its bandwidth to 2 Gbps. This causes an immediate drop in training performance and triggers Autotune at step=3. Quicktune moves LMs away from node 0. The training throughput then bumps up immediately, and Autotune enters fine-tuning mode through step=11. During this period, Autotune continues to move LM assignments away from 0. When 0 is out of LMs, Finetune moves LMs among the rest of the nodes based on their blame score. At step=11, node 0's bandwidth is restored, causing an immediate jump in training performance, but this time node 0 is underloaded. At step=12, this is partially corrected by Quicktune. Autotune continues to monitor and rebalance workloads throughout, arriving at a balanced assignment at t=25, with $\frac{max_n \sum_i D(n,i)}{min_n \sum_i D(n,i)} < 1.05$. Compared to using this near optimally balanced LM assignment alone, Autotune delivers 1.27x speedup when node 0 is the bottleneck, and can quickly recover when node 0's limit is lifted.

6 Related Work and Discussion

Network topology discovery. We use similar embedding approach to (Dabek et al., 2004) with a different goal: they embed peers on the globe, whereas we use embedding to denoise ProbeEmbed probes with a novel objective.

(Battre et al., 2011) provides a technique for topology inference. In contrast, we are interested in clustering based on locality that encompasses both topology and dynamic load. Further, (Battre et al., 2011) explored using jitter and loss experienced by train-like (Hu & Steenkiste, 2003) probes to infer topology, but acknowledged that it was ineffective in a fully virtualized environment (Section III.C), the setting that we target (public clouds). We confirmed such methods for locality discovery performed poorly given the low latency of modern switches; jitter introduced by queueing is too small to provide meaningful measurements. Lastly, (Lawrence & Yuan, 2008) assumes store-and-forward switches in homogeneous clusters forming trees without virtualization.

Cloud-aware MPI. We share some similarities in terms of using measurements to guide a hierarchical aggregation mechanism used in (Gong et al., 2015; Alfatafta et al., 2018). Our work differs in the following ways: (1) ProbeEmbed includes denoising techniques to provide accurate embeddings in a noisy cloud environment, and we morph reduction trees to adapt to dynamic load. (2) We target allreduce operations as opposed to the simpler reduce/scatter/gather primitives considered by (Gong et al., 2015). (3) We exploit locality both in the network and within each node, through the design of AggEngine. As a result, we achieve better performance at a larger scale than (Gong et al., 2015) (which achieved 13-27% speedup on 32 EC2 nodes). (4) We demonstrate actual end-to-end application-level speedups by integrating with ML frameworks for real world workloads, while (Gong et al., 2015) ran microbenchmarks and (Alfatafta et al., 2018) did an analytical evaluation.

Cross region training. Many (Cano et al., 2016; Hsieh et al., 2017) have explored the possibility of geo-distributed learning. PLink can help here both by reducing traffic through bottleneck links (inter-datacenter, inter-availability zone) and by reducing the cost of training as cloud providers usually charge for cross-region/zone transfers.

Large batch optimization. Large batch sizes reduce communication frequency (FastAI; Goyal et al., 2017; Sridharan et al., 2018; Jia et al., 2018a), but also eliminates the potential of achieving a larger speedup with a fast communication plane: with ResNet-50, (Shen et al., 2019) shows only 10 samples are needed to utilize a recent GPU fully, and thus the computation of large batches can be further spread to more GPUs, provided that communication overhead is low.

Quantization, compression, and matrix decomposition. Communication bottlenecks can be alleviated by trading potentially more iterations for fewer bytes sent per iteration, with sigma-delta modulation (Seide et al., 2014), or fully-connected layer decomposition (Zhang et al., 2017; Chilimbi et al., 2014), or with redundancy reduction in SGD (Lin et al., 2017). These techniques push the communication bottleneck towards latency and work well with PLink to take advantage of its lower latency stack. §5.3 suggests in the extreme case of a single-bit reduction, PLink still achieves meaningful speedup; comparing Figure 6 with Figure 1, PLink boosts the training throughput of Pytorch/Caffe2 with full gradient precision to be similar to MxNet with gradient compression and 2-bit quantization.

Reinforcement and active learning. While past work has shown promising results on the use of reinforcement learning on scheduling and placement (Chen et al., 2018b;a; Kraska et al., 2018; Mirhoseini et al., 2017; Bodin et al., 2018), our preliminary assessment of end-to-end RL for PLink is unfavourable. RL requires a large samples covering all conditions, and collecting training throughput on the public cloud has prohibitive costs. Further, an agent is unlikely to see a significant fraction of the possible topologies with a constantly evolving cloud. However, a coordinated effort by cloud providers across customer jobs may be feasible as it could train on more samples with greater diversity.

Predicting bandwidth changes. Autotune currently reacts to bandwidth changes only after they happen. This can be improved by carefully monitoring connection state variables such as the congestion window to anticipate an incoming bandwidth change (Haeri & Rad, 2006; Biaz & Vaidya, 1998; Kong et al., 2018; Li et al., 2016).

Scheduling for locality. Some cluster schedulers make placement decisions that consider network locality (Xiao et al., 2018b; Gu et al., 2019; Jia et al., 2018b), but this can increase scheduling time. Autotune is complimentary in that it can provide better throughput when good placements are not possible.

7 Conclusion

Accelerating distributed training in the public cloud has unique challenges, making communication a bottleneck in the training process. We proposed PLink, a system that accurately probes the network and efficiently generates and executes topology-aware, hierarchical aggregation plans, taking advantage of the locality in datacenter networks, and continuously balances workloads across VM nodes to adapt to changing network conditions. We show that PLink achieves an end-to-end training speedup in unmodified commercial clouds of up to 2.3x with popular neural network models.

Acknowledgements

This work was supported in part by NSF award CNS-1614717 and CCF-1518703; by CRISP, one of six centers in JUMP, a Semiconductor Research Corporation (SRC) program sponsored by DARPA; by gifts from Xilinx, Intel (under the CAPA program), Oracle, Amazon, Qualcomm, Facebook, Futurewei, Google, and other anonymous sources.

REFERENCES

Akiba, T., Suzuki, S., and Fukuda, K. Extremely large minibatch SGD: training resnet-50 on imagenet in 15 minutes. CoRR, abs/1711.04325, 2017. URL http://arxiv.org/abs/1711.04325.

Alfatafta, M., AlSader, Z., and Al-Kiswany, S. Cool: A cloud-optimized structure for mpi collective operations. In 2018 IEEE 11th International Conference on Cloud Computing (CLOUD), pp. 746–753, July 2018. doi: 10.1109/CLOUD.2018.00102.

Alizadeh, M., Greenberg, A., Maltz, D. A., Padhye, J., Patel, P., Prabhakar, B., Sengupta, S., and Sridharan, M. Data center tcp (dctcp). SIGCOMM Comput. Commun. Rev., 40(4):6374, August 2010. ISSN 0146-4833. doi: 10.1145/1851275.1851192. URL https://doi.org/10.1145/1851275.1851192.

Amazon. Deep learning on aws. https://aws.amazon.com/deep-learning/, a. (Accessed on 12/06/2018).

Amazon. Enable and configure enhanced networking for ec2 instances. https://aws.amazon.com/premiumsupport/knowledge-center/enable-configure-enhanced-networking/, b. (Accessed on 01/06/2019).

Amazon. Introducing dynamic training for deep learning with amazon ec2 — aws machine learning blog. https://aws.amazon.com/blogs/machine-learning/introducing-dynamic-training-for-deep-learning-with-amazon-ec2/, c. (Accessed on 12/07/2018).

Amazon. Placement groups - amazon elastic compute cloud. https://docs.aws.amazon.com/AWSEC2/latest/UserGuide/placement-groups.html, d. (Accessed on 12/07/2018).

Amazon. Amazon ec2 f1 instances. https://aws.amazon.com/ec2/instance-types/f1/, e. (Accessed on 03/04/2019).

Arthur, D. and Vassilvitskii, S. k-means++: The advantages of careful seeding. In Proceedings of the eighteenth annual ACM-SIAM symposium on Discrete algorithms, pp. 1027–1035. Society for Industrial and Applied Mathematics, 2007.

Bala, V., Bruck, J., Cypher, R., Elustondo, P., Ho, A., Ho, C.-T., Kipnis, S., and Snir, M. Ccl: A portable and tunable collective communication library for scalable parallel computers. IEEE Transactions on Parallel and Distributed Systems, 6(2):154–164, 1995.

Battre, D., Frejnik, N., Goel, S., Kao, O., and Warneke, D. Evaluation of network topology inference in opaque compute clouds through end-to-end measurements. In 2011 IEEE 4th International Conference on Cloud Computing, pp. 17–24, July 2011. doi: 10.1109/CLOUD.2011.30.

Biaz, S. and Vaidya, N. H. Performance of tcp congestion predictors as loss predictors. Department of Computer Science Technical Report, 98(7), 1998.

Bilal, K., Khan, S. U., Kolodziej, J., Zhang, L., Hayat, K., Madani, S. A., Min-Allah, N., Wang, L., and Chen, D. A comparative study of data center network architectures. In ECMS, 2012.

Blum, E. K., Wang, X., and Leung, P. Architectures and message-passing algorithms for cluster computing: Design and performance. Parallel Computing, 26(2-3):313–332, 2000.

Bodin, B., Nardi, L., Wagstaff, H., Kelly, P. H. J., and O'Boyle, M. Algorithmic performance-accuracy trade-off in 3d vision applications. In 2018 IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS), pp. 123–124, April 2018. doi: 10.1109/ISPASS.2018.00024.

Bradley, P., Bennett, K., and Demiriz, A. Constrained k-means clustering. Microsoft Research, Redmond, 20(0):0, 2000.

Cano, I., Weimer, M., Mahajan, D., Curino, C., and Fumarola, G. M. Towards geo-distributed machine learning, 2016.

Chen, T., Moreau, T., Jiang, Z., Zheng, L., Yan, E., Shen, H., Cowan, M., Wang, L., Hu, Y., Ceze, L., Guestrin, C., and Krishnamurthy, A. TVM: An automated end-to-end optimizing compiler for deep learning. In 13th USENIX Symposium on Operating Systems Design and Implementation (OSDI 18), pp. 578–594, Carlsbad, CA, 2018a. USENIX Association. ISBN 978-1-931971-47-8. URL https://www.usenix.org/conference/osdi18/presentation/chen.

Chen, T., Zheng, L., Yan, E., Jiang, Z., Moreau, T., Ceze, L., Guestrin, C., and Krishnamurthy, A. Learning to optimize tensor programs. arXiv preprint arXiv:1805.08166, 2018b.

Chilimbi, T., Suzue, Y., Apacible, J., and Kalyanaraman, K. Project adam: Building an efficient and scalable deep learning training system. In 11th USENIX Symposium on Operating Systems Design and Implementation (OSDI 14), pp. 571–582, Broomfield, CO, 2014. USENIX Association. ISBN 978-1-931971-16-4. URL https://www.usenix.org/

conference/osdi14/technical-sessions/presentation/chilimbi.

Cho, M., Finkler, U., and Kung, D. Blueconnect: Novel hierarchical all-reduce on multi-tired network for deep learning.

Cui, H., Zhang, H., Ganger, G. R., Gibbons, P. B., and Xing, E. P. Geeps: Scalable deep learning on distributed gpus with a gpu-specialized parameter server. In Proceedings of the Eleventh European Conference on Computer Systems, pp. 4. ACM, 2016.

Dabek, F., Cox, R., Kaashoek, F., and Morris, R. Vivaldi: A decentralized network coordinate system. SIGCOMM Comput. Commun. Rev., 34(4):15–26, August 2004. ISSN 0146-4833. doi: 10.1145/1030194.1015471. URL http://doi.acm.org/10.1145/1030194.1015471.

dpdk. Home - dpdk. https://www.dpdk.org/. (Accessed on 09/01/2019).

Facebook. facebookincubator/gloo: Collective communications library with various primitives for multi-machine training. https://github.com/facebookincubator/gloo. (Accessed on 12/22/2018).

Farley, B., Juels, A., Varadarajan, V., Ristenpart, T., Bowers, K. D., and Swift, M. M. More for your money: Exploiting performance heterogeneity in public clouds. In Proceedings of the Third ACM Symposium on Cloud Computing, SoCC '12, pp. 20:1–20:14, New York, NY, USA, 2012. ACM. ISBN 978-1-4503-1761-0. doi: 10.1145/2391229.2391249. URL http://doi.acm.org/10.1145/2391229.2391249.

FastAI. Now anyone can train imagenet in 18 minutes fast.ai. https://www.fast.ai/2018/08/10/fastai-diu-imagenet/. (Accessed on 12/20/2018).

Fowers, J., Ovtcharov, K., Papamichael, M., Massengill, T., Liu, M., Lo, D., Alkalay, S., Haselman, M., Adams, L., Ghandi, M., et al. A configurable cloud-scale dnn processor for real-time ai. In Proceedings of the 45th Annual International Symposium on Computer Architecture, pp. 1–14. IEEE Press, 2018.

Geng, J., Li, D., Cheng, Y., Wang, S., and Li, J. Hips: Hierarchical parameter synchronization in large-scale distributed machine learning. In Proceedings of the 2018 Workshop on Network Meets AI and ML, NetAI'18, pp. 1–7, New York, NY, USA, 2018. ACM. ISBN 978-1-4503-5911-5. doi: 10.1145/3229543.3229544. URL http://doi.acm.org/10.1145/3229543.3229544.

Gong, Y., He, B., and Zhong, J. Network performance aware mpi collective communication operations in the cloud. IEEE Transactions on Parallel and Distributed Systems, 26(11):3079–3089, Nov 2015. ISSN 2161-9883. doi: 10.1109/TPDS.2013.96.

Google. Google cloud training. google cloud. https://cloud.google.com/training/courses/machine-learning-tensorflow-gcp. (Accessed on 03/04/2019).

Goyal, P., Dollár, P., Girshick, R., Noordhuis, P., Wesolowski, L., Kyrola, A., Tulloch, A., Jia, Y., and He, K. Accurate, large minibatch SGD: Training ImageNet in 1 hour. arXiv preprint arXiv:1706.02677, 2017.

Graham, R. L., Bureddy, D., Lui, P., Rosenstock, H., Shainer, G., Bloch, G., Goldenerg, D., Dubman, M., Kotchubievsky, S., Koushnir, V., Levi, L., Margolin, A., Ronen, T., Shpiner, A., Wertheim, O., and Zahavi, E. Scalable hierarchical aggregation protocol (sharp): A hardware architecture for efficient data reduction. In Proceedings of the First Workshop on Optimization of Communication in HPC, COM-HPC '16, pp. 1–10, Piscataway, NJ, USA, 2016. IEEE Press. ISBN 978-1-5090-3829-9. doi: 10.1109/COM-HPC.2016.6. URL https://doi.org/10.1109/COM-HPC.2016.6.

Greenberg, A., Hamilton, J. R., Jain, N., Kandula, S., Kim, C., Lahiri, P., Maltz, D. A., Patel, P., and Sengupta, S. Vl2: A scalable and flexible data center network. SIGCOMM Comput. Commun. Rev., 39 (4):5162, August 2009. ISSN 0146-4833. doi: 10.1145/1594977.1592576. URL https://doi.org/10.1145/1594977.1592576.

Gu, J., Chowdhury, M., Shin, K. G., Zhu, Y., Jeon, M., Qian, J., Liu, H., and Guo, C. Tiresias: A GPU cluster manager for distributed deep learning. In 16th USENIX Symposium on Networked Systems Design and Implementation (NSDI 19), pp. 485–500, Boston, MA, February 2019. USENIX Association. ISBN 978-1-931971-49-2. URL https://www.usenix.org/conference/nsdi19/presentation/gu.

Haeri, M. and Rad, A. H. M. Adaptive model predictive tcp delay-based congestion control. Computer Communications, 29(11):1963–1978, 2006.

Hashemi, S. H., Jyothi, S. A., and Campbell, R. H. Tictac: Accelerating distributed deep learning with communication scheduling. arXiv preprint arXiv:1803.03288, 2018.

Hsieh, K., Harlap, A., Vijaykumar, N., Konomis, D., Ganger, G. R., Gibbons, P. B., and Mutlu, O. Gaia: Geo-distributed machine learning approaching LAN speeds. In 14th USENIX Symposium on Networked Systems

Design and Implementation (NSDI 17), pp. 629–647, Boston, MA, 2017. USENIX Association. ISBN 978-1-931971-37-9. URL https://www.usenix.org/conference/nsdi17/technical-sessions/presentation/hsieh.

Hu, N. and Steenkiste, P. Evaluation and characterization of available bandwidth probing techniques. IEEE Journal on Selected Areas in Communications, 21(6):879–894, Aug 2003. ISSN 0733-8716. doi: 10.1109/JSAC.2003.814505.

Iandola, F. N., Moskewicz, M. W., Ashraf, K., and Keutzer, K. Firecaffe: near-linear acceleration of deep neural network training on compute clusters. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, pp. 2592–2600, 2016.

IBM. Softiwarp for linux-rdma. https://github.com/zrlio/softiwarp-for-linux-rdma. (Accessed on 11/06/2019).

Iosup, A., Yigitbasi, N., and Epema, D. On the performance variability of production cloud services. In Proceedings of the 2011 11th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing, CCGRID '11, pp. 104–113, Washington, DC, USA, 2011. IEEE Computer Society. ISBN 978-0-7695-4395-6. doi: 10.1109/CCGrid.2011.22. URL http://dx.doi.org/10.1109/CCGrid.2011.22.

iperf. iperf - the tcp, udp and sctp network bandwidth measurement tool. https://iperf.fr/. (Accessed on 09/01/2019).

Jayarajan, A., Wei, J., Gibson, G., Fedorova, A., and Pekhimenko, G. Priority-based parameter propagation for distributed dnn training. arXiv preprint arXiv:1905.03960, 2019.

Jeyakumar, V., Alizadeh, M., Mazières, D., Prabhakar, B., and Kim, C. Eyeq: Practical network performance isolation for the multi-tenant cloud. In HotCloud, 2012.

Jia, X., Song, S., He, W., Wang, Y., Rong, H., Zhou, F., Xie, L., Guo, Z., Yang, Y., Yu, L., Chen, T., Hu, G., Shi, S., and Chu, X. Highly scalable deep learning training system with mixed-precision: Training imagenet in four minutes, 2018a.

Jia, Z., Zaharia, M., and Aiken, A. Beyond data and model parallelism for deep neural networks. arXiv preprint arXiv:1807.05358, 2018b.

Jouppi, N. P., Young, C., Patil, N., Patterson, D., Agrawal, G., Bajwa, R., Bates, S., Bhatia, S., Boden, N., Borchers, A., Boyle, R., Cantin, P.-l., Chao, C., Clark, C., Coriell, J., Daley, M., Dau, M., Dean, J., Gelb, B., Ghaemmaghami, T. V., Gottipati, R., Gulland, W., Hagmann, R., Ho, C. R., Hogberg, D., Hu, J., Hundt, R., Hurt, D., Ibarz, J., Jaffey, A., Jaworski, A., Kaplan, A., Khaitan, H., Killebrew, D., Koch, A., Kumar, N., Lacy, S., Laudon, J., Law, J., Le, D., Leary, C., Liu, Z., Lucke, K., Lundin, A., MacKean, G., Maggiore, A., Mahony, M., Miller, K., Nagarajan, R., Narayanaswami, R., Ni, R., Nix, K., Norrie, T., Omernick, M., Penukonda, N., Phelps, A., Ross, J., Ross, M., Salek, A., Samadiani, E., Severn, C., Sizikov, G., Snelham, M., Souter, J., Steinberg, D., Swing, A., Tan, M., Thorson, G., Tian, B., Toma, H., Tuttle, E., Vasudevan, V., Walter, R., Wang, W., Wilcox, E., and Yoon, D. H. In-datacenter performance analysis of a tensor processing unit. In Proceedings of the 44th Annual International Symposium on Computer Architecture, ISCA '17, pp. 1–12, New York, NY, USA, 2017. ACM. ISBN 978-1-4503-4892-8. doi: 10.1145/3079856.3080246. URL http://doi.acm.org/10.1145/3079856.3080246.

Kingma, D. P. and Ba, J. Adam: A method for stochastic optimization. arXiv preprint arXiv:1412.6980, 2014.

Kong, Y., Zang, H., and Ma, X. Improving tcp congestion control with machine intelligence. In Proceedings of the 2018 Workshop on Network Meets AI & ML, pp. 60–66. ACM, 2018.

Kraska, T., Beutel, A., Chi, E. H., Dean, J., and Polyzotis, N. The case for learned index structures. In Proceedings of the 2018 International Conference on Management of Data, SIGMOD '18, pp. 489–504, New York, NY, USA, 2018. ACM. ISBN 978-1-4503-4703-7. doi: 10.1145/3183713.3196909. URL http://doi.acm.org/10.1145/3183713.3196909.

Lawrence, J. and Yuan, X. An mpi tool for automatically discovering the switch level topologies of ethernet clusters. In 2008 IEEE International Symposium on Parallel and Distributed Processing, pp. 1–8, April 2008. doi: 10.1109/IPDPS.2008.4536545.

Li, M., Andersen, D. G., Park, J. W., Smola, A. J., Ahmed, A., Josifovski, V., Long, J., Shekita, E. J., and Su, B.-Y. Scaling distributed machine learning with the parameter server. In Proceedings of the 11th USENIX Conference on Operating Systems Design and Implementation, OSDI'14, pp. 583–598, Berkeley, CA, USA, 2014a. USENIX Association. ISBN 978-1-931971-16-4. URL http://dl.acm.org/citation.cfm?id=2685048.2685095.

Li, M., Andersen, D. G., Smola, A., and Yu, K. Communication efficient distributed machine learning with the parameter server. In Proceedings of the 27th International Conference on Neural Information Processing Systems, NIPS'14, pp. 19–27, Cambridge, MA, USA, 2014b.

MIT Press. URL http://dl.acm.org/citation.cfm?id=2968826.2968829.

Li, W., Zhou, F., Meleis, W., and Chowdhury, K. Learning-based and data-driven tcp design for memory-constrained iot. In Distributed Computing in Sensor Systems (DCOSS), 2016 International Conference on, pp. 199–205. IEEE, 2016.

Lim, H., Andersen, D. G., and Kaminsky, M. 3lc: Lightweight and effective traffic compression for distributed machine learning. arXiv preprint arXiv:1802.07389, 2018.

Lin, Y., Han, S., Mao, H., Wang, Y., and Dally, W. J. Deep gradient compression: Reducing the communication bandwidth for distributed training. arXiv preprint arXiv:1712.01887, 2017.

Liu, M., Luo, L., Nelson, J., Ceze, L., Krishnamurthy, A., and Atreya, K. IncBricks: Toward in-network computation with an in-network cache. SIGOPS Oper. Syst. Rev., 51(2):795–809, April 2017. ISSN 0163-5980. doi: 10.1145/3093315.3037731. URL http://doi.acm.org/10.1145/3093315.3037731.

Luo, L., Liu, M., Nelson, J., Ceze, L., Phanishayee, A., and Krishnamurthy, A. Motivating in-network aggregation for distributed deep neural network training. In Workshop on Approximate Computing Across the Stack, 2017.

Luo, L., Nelson, J., Ceze, L., Phanishayee, A., and Krishnamurthy, A. Parameter hub: A rack-scale parameter server for distributed deep neural network training. In Proceedings of the ACM Symposium on Cloud Computing, SoCC '18, pp. 41–54, New York, NY, USA, 2018. ACM. ISBN 978-1-4503-6011-1. doi: 10.1145/3267809.3267840. URL http://doi.acm.org/10.1145/3267809.3267840.

Maricq, A., Duplyakin, D., Jimenez, I., Maltzahn, C., Stutsman, R., and Ricci, R. Taming performance variability. In 13th USENIX Symposium on Operating Systems Design and Implementation (OSDI 18), pp. 409–425, Carlsbad, CA, 2018. USENIX Association. ISBN 978-1-931971-47-8. URL https://www.usenix.org/conference/osdi18/presentation/maricq.

Mathis, M., Heffner, J., and Reddy, R. Web100: extended tcp instrumentation for research, education and diagnosis. ACM SIGCOMM Computer Communication Review, 33(3):69–79, 2003.

Metzler, B., Frey, P., and Trivedi, A. Softiwarp project update: A software iwarp driver for openfabrics. 03 2010. doi: 10.13140/2.1.4422.3363.

Microsoft. Azure batch ai — train ai models — microsoft azure. https://azure.microsoft.com/en-us/services/batch-ai/, a. (Accessed on 12/07/2018).

Microsoft. Create an azure virtual machine with accelerated networking — microsoft docs. https://docs.microsoft.com/en-us/azure/virtual-network/create-vm-accelerated-networking-cli, b. (Accessed on 01/06/2019).

Microsoft. Machine learning studio. microsoft azure. https://azure.microsoft.com/en-us/services/machine-learning-studio/, c. (Accessed on 12/06/2018).

Mikami, H., Suganuma, H., U.-Chupala, P., Tanaka, Y., and Kageyama, Y. Imagenet/resnet-50 training in 224 seconds. CoRR, abs/1811.05233, 2018. URL http://arxiv.org/abs/1811.05233.

Mirhoseini, A., Pham, H., Le, Q. V., Steiner, B., Larsen, R., Zhou, Y., Kumar, N., Norouzi, M., Bengio, S., and Dean, J. Device placement optimization with reinforcement learning. arXiv preprint arXiv:1706.04972, 2017.

mixpanel. 27 — october — 2011 — mixpanel engineering. https://engineering.mixpanel.com/2011/10/27/. (Accessed on 12/07/2018).

Mysore, R. N., Pamboris, A., Farrington, N., Huang, N., Miri, P., Radhakrishnan, S., Subramanya, V., and Vahdat, A. Portland: a scalable fault-tolerant layer 2 data center network fabric. In SIGCOMM, 2009.

Nvidia. Operations - nccl 2.3.4 documentation. https://docs.nvidia.com/deeplearning/sdk/nccl-developer-guide/docs/usage/operations.html. (Accessed on 12/07/2018).

Pesterev, A., Strauss, J., Zeldovich, N., and Morris, R. T. Improving network connection locality on multicore systems. In Proceedings of the 7th ACM European Conference on Computer Systems, EuroSys '12, pp. 337–350, New York, NY, USA, 2012. ACM. ISBN 978-1-4503-1223-3. doi: 10.1145/2168836.2168870. URL http://doi.acm.org/10.1145/2168836.2168870.

PSLite. dmlc/ps-lite: A lightweight parameter server interface. https://github.com/dmlc/ps-lite. (Accessed on 12/22/2018).

Rabenseifner, R. Optimization of collective reduction operations. pp. 1–9, 06 2004. doi: 10.1007/978-3-540-24685-5_1.

Roy, A., Zeng, H., Bagga, J., Porter, G., and Snoeren, A. C. Inside the social network's (datacenter) network. Computer Communication Review, 45:123–137, 2015.

Sack, P. D. Scalable Collective Message-passing Algorithms. PhD thesis, Champaign, IL, USA, 2011. AAI3503864.

scikit learn. 4.8. pairwise metrics, affinities and kernels. scikit-learn 0.20.2 documentation. https://scikit-learn.org/stable/modules/metrics.html. (Accessed on 02/06/2019).

Seide, F., Fu, H., Droppo, J., Li, G., and Yu, D. 1-bit stochastic gradient descent and application to data-parallel distributed training of speech DNNs. In Interspeech 2014, September 2014.

Sergeev, A. and Balso, M. D. Horovod: fast and easy distributed deep learning in tensorflow. CoRR, abs/1802.05799, 2018.

Shen, H., Chen, L., Jin, Y., Zhao, L., Kong, B., Philipose, M., Krishnamurthy, A., and Sundaram, R. Nexus: A gpu cluster engine for accelerating dnn-based video analysis. In Proceedings of the 27th ACM Symposium on Operating Systems Principles, SOSP 19, pp. 322337, New York, NY, USA, 2019. Association for Computing Machinery. ISBN 9781450368735. doi: 10.1145/3341301.3359658. URL https://doi.org/10.1145/3341301.3359658.

Shieh, A., Kandula, S., Greenberg, A., and Kim, C. Seawall: Performance isolation for cloud datacenter networks. In Proceedings of the 2Nd USENIX Conference on Hot Topics in Cloud Computing, HotCloud'10, pp. 1–1, Berkeley, CA, USA, 2010. USENIX Association. URL http://dl.acm.org/citation.cfm?id=1863103.1863104.

Shieh, A., Kandula, S., Greenberg, A., Kim, C., and Saha, B. Sharing the data center network. In Proceedings of the 8th USENIX Conference on Networked Systems Design and Implementation, NSDI'11, pp. 309–322, Berkeley, CA, USA, 2011. USENIX Association. URL http://dl.acm.org/citation.cfm?id=1972457.1972489.

Smith, S. L., Kindermans, P., and Le, Q. V. Don't decay the learning rate, increase the batch size. CoRR, abs/1711.00489, 2017. URL http://arxiv.org/abs/1711.00489.

Smola, A. and Narayanamurthy, S. An architecture for parallel topic models. Proc. VLDB Endow., 3(1-2):703–710, September 2010. ISSN 2150-8097. doi: 10.14778/1920841.1920931. URL http://dx.doi.org/10.14778/1920841.1920931.

SoftRoCE. Soft rdma over ethernet (roce) driver. https://github.com/SoftRoCE/rxe-dev/wiki/rxe-dev:-Home. (Accessed on 12/16/2018).

Sridharan, S., Vaidyanathan, K., Kalamkar, D., Das, D., Smorkalov, M. E., Shiryaev, M., Mudigere, D., Mellempudi, N., Avancha, S., Kaul, B., and Dubey, P. On scale-out deep learning training for cloud and hpc, 2018.

Sun, P., Feng, W., Han, R., Yan, S., and Wen, Y. Optimizing network performance for distributed dnn training on gpu clusters: Imagenet/alexnet training in 1.5 minutes. arXiv preprint arXiv:1902.06855, 2019.

Thakur, R., Rabenseifner, R., and Gropp, W. Optimization of collective communication operations in mpich. Int. J. High Perform. Comput. Appl., 19(1):49–66, February 2005. ISSN 1094-3420. doi: 10.1177/1094342005051521. URL http://dx.doi.org/10.1177/1094342005051521.

Vinh, N. X., Epps, J., and Bailey, J. Information theoretic measures for clusterings comparison: Variants, properties, normalization and correction for chance. Journal of Machine Learning Research, 11(Oct):2837–2854, 2010.

Wangt, G., Phanishayee, A., Venkataraman, S., and Stoicat, I. Blink: A fast nvlink-based collective communication library. 2018.

Wikipedia. Netlink wikipedia. https://en.wikipedia.org/wiki/Netlink#cite_note-4. (Accessed on 12/22/2018).

Xiao, W., Bhardwaj, R., Ramjee, R., Sivathanu, M., Kwatra, N., Han, Z., Patel, P., Peng, X., Zhao, H., Zhang, Q., Yang, F., and Zhou, L. Gandiva: Introspective cluster scheduling for deep learning. In 13th USENIX Symposium on Operating Systems Design and Implementation (OSDI 18), pp. 595–610, Carlsbad, CA, 2018a. USENIX Association. ISBN 978-1-931971-47-8. URL https://www.usenix.org/conference/osdi18/presentation/xiao.

Xiao, W., Bhardwaj, R., Ramjee, R., Sivathanu, M., Kwatra, N., Han, Z., Patel, P., Peng, X., Zhao, H., Zhang, Q., Yang, F., and Zhou, L. Gandiva: Introspective cluster scheduling for deep learning. In 13th USENIX Symposium on Operating Systems Design and Implementation (OSDI 18), pp. 595–610, Carlsbad, CA, October 2018b. USENIX Association. ISBN 978-1-939133-08-3. URL https://www.usenix.org/conference/osdi18/presentation/xiao.

You, Y., Zhang, Z., Hsieh, C.-J., Demmel, J., and Keutzer, K. Imagenet training in minutes. In Proceedings of the 47th International Conference on Parallel Processing, ICPP 2018, pp. 1:1–1:10, New York, NY, USA, 2018. ACM. ISBN 978-1-4503-6510-9. doi: 10.1145/3225058.3225069. URL `http://doi.acm.org/10.1145/3225058.3225069`.

Zhang, C. and Ré, C. Dimmwitted: A study of main-memory statistical analytics. Proc. VLDB Endow., 7 (12):1283–1294, August 2014. ISSN 2150-8097. doi: 10.14778/2732977.2733001. URL `http://dx.doi.org/10.14778/2732977.2733001`.

Zhang, H., Zheng, Z., Xu, S., Dai, W., Ho, Q., Liang, X., Hu, Z., Wei, J., Xie, P., and Xing, E. P. Poseidon: An efficient communication architecture for distributed deep learning on GPU clusters. In 2017 USENIX Annual Technical Conference (USENIX ATC 17), pp. 181–193, Santa Clara, CA, 2017. USENIX Association. ISBN 978-1-931971-38-6. URL `https://www.usenix.org/conference/atc17/technical-sessions/presentation/zhang`.