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

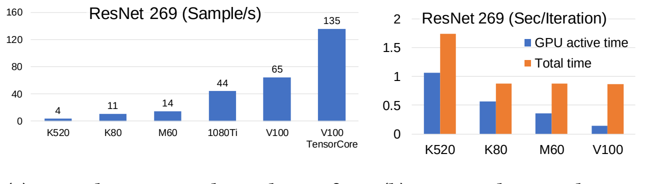
Most work in the deep learning systems community has focused on faster inference, but arriving at a trained model requires lengthy experiments. Accelerating training lets developers iterate faster and come up with better models.

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.*



(a) Per-chip GPU throughput for ResNet 269 training in EC2 has increased by 35x since 2012.

(b) Increased network overhead in training as GPUs get faster.

Figure 1: Distributed training as a communication bound workload in the cloud.

Framework	Local	2 workers	4 workers	8 workers
TensorFlow	152	213	410	634
Caffe2	195	266	343	513
MXNet	190	187	375	**688**

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).

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.

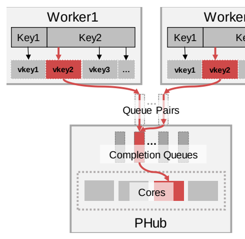
Figure 1b summarizes the throughput of modest-scale DNN training with 8 workers and 8 colocated PSs on EC2 with 10Gbps links and a per GPU batch size of 4 (maximizing GPU memory usage on GRID 520): although modern DNN training frameworks can overlap backward passes with model updates, they can no longer 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.

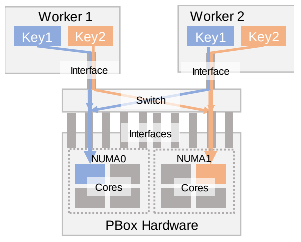
These observations suggest that DDNN training has shifted from a compute-bound problem to one that also has a significant network-bound component. It is critical to perform model updates efficiently.

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



(a) Fine grained key chunking and balanced chunk to core assignment scheme in PHub.



(b) PBox has multiple NICs to balance IO, memory and network bandwidth.

Figure 2: The PHub software and hardware architecture

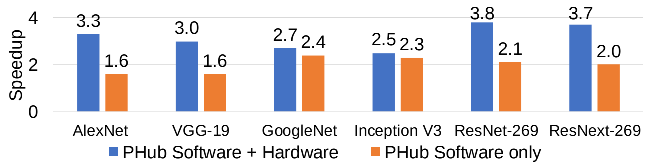


Figure 3: Training performance on a EC2-like 10Gbps network. Results are normalized to sharded MXNet. Batch size per GPU: 8 for ResNext, 16 for ResNet, 32 for others. GPU: GTX 1080 Ti. Higher speedup possible with latest GPUs.

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 single box. We integrated PHub with MXNet; Figure 3 shows the speedup of PHub over a MXNet colocated sharded baseline when training ImageNet-winning networks with 8 workers. PHub on PBox can provide up to 3.8x speedup over the state-of-the-art on a cloud-like 10Gbps network[3]. The speedup using 56Gbps links is similar, ranging from 2x-7x depending on the DNN being trained.

[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.

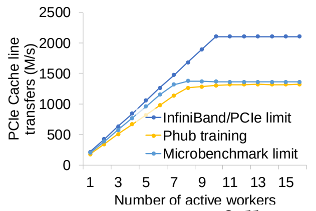


Figure 4: PBox can fully utilize hardware during training. Its performance is bottlenecked by the PCIe/cache controller link.

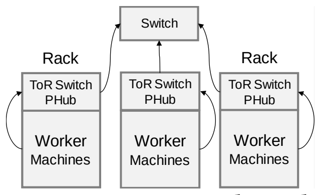


Figure 5: A potential PHub implementation with a programmable ToR switch with hybrid aggregation for workers in different racks.

PHub shows linear scaling with our 8 worker cluster running all workloads, and provides higher throughput than other parameter exchange patterns using MPI or collectives (because PHub uses only one round of communication and minimum total data transfer per iteration). To understand its limits, we used a special `ZeroComputeEngine` that simulates infinitely fast computation in MXNet, performing only parameter exchange operations. We found PBox performance is limited only by the bandwidth between the PCIe controller and the memory system (Figure 4). This limit is hard to hit in real training: we estimate that a single PBox will support up to 120 workers training ResNet-50 with batch size of 32 per GPU. If each worker includes 4 GPUs, that translates to a global batch size of 15K, surpassing the maximum suggested in [6]. Recent work suggests larger batch sizes may impede training [3, 6–8], but if higher scalability is desired, sharding or use of new platforms with more PCIe throughput (e.g., [1]) would enable PHub to provide higher throughput.

3 IN-NETWORK AGGREGATION

The PBox results show the benefit of a high-bandwidth centralized PS. Recent programmable switches [9, 12, 13] enable a new approach to building centralized PS designs by allowing gradient aggregation operations to be performed in the network. Figure 5 shows how PHub running in a top of rack (ToR) switch can benefit from the full bisection bandwidth inside a server rack, performing centralized aggregation of gradients inside a rack; only a single aggregated stream must be sent to higher level switches for further aggregation across racks. This hybrid synchronization reduces bandwidth usage and localizes data movement in the data center.

Current switches have limited computational capabilities: most can perform only integer operations, with little on-switch storage, and only on a small region of each packet. Our future work includes exploring the hardware requirements necessary for efficient DDNN training, as our emulation-based experiments show that these limits lead to unsatisfactory throughput on current switches.

[3]All interfaces have a negotiated speed of 10Gbps with the switch in this experiment.

REFERENCES

[1] [n. d.]. AMD EPYC. http://www.amd.com/en/products/epyc. ([n. d.]).

[2] [n. d.]. EC2Instances.info Easy Amazon EC2 Instance Comparison. https://www.ec2instances.info/?region=us-west-2. ([n. d.]).

[3] [n. d.]. IBM Research achieves record deep learning performance with new software technology. https://www.ibm.com/blogs/research/2017/08/distributed-deep-learning/. ([n. d.]).

[4] [n. d.]. Performance of Distributed Deep Learning using ChainerMN. https://chainer.org/general/2017/02/08/Performance-of-Distributed-Deep-Learning-Using-ChainerMN.html. ([n. d.]).

[5] Tianqi Chen, Bing Xu, Chiyuan Zhang, and Carlos Guestrin. 2016. Training deep nets with sublinear memory cost. *arXiv preprint arXiv:1604.06174* (2016).

[6] Priya Goyal, Piotr Dollár, Ross Girshick, Pieter Noordhuis, Lukasz Wesolowski, Aapo Kyrola, Andrew Tulloch, Yangqing Jia, and Kaiming He. 2017. Accurate, Large Minibatch SGD: Training ImageNet in 1 Hour. *arXiv preprint arXiv:1706.02677* (2017).

[7] Nitish Shirish Keskar, Dheevatsa Mudigere, Jorge Nocedal, Mikhail Smelyanskiy, and Ping Tak Peter Tang. 2016. On large-batch training for deep learning: Generalization gap and sharp minima. *arXiv preprint arXiv:1609.04836* (2016).

[8] Y LeCun, L Bottou, and G Orr. [n. d.]. Efficient BackProp in Neural Networks: Tricks of the Trade (Orr, G. and Müller, K., eds.). *Lecture Notes in Computer Science* 1524 ([n. d.]).

[9] Jialin Li, Ellis Michael, Naveen Kr. Sharma, Adriana Szekeres, and Dan R. K. Ports. 2016. Just Say No to Paxos Overhead: Replacing Consensus with Network Ordering. In *Proceedings of the 12th USENIX Conference on Operating Systems Design and Implementation (OSDI'16)*. USENIX Association, Berkeley, CA, USA, 467–483. http://dl.acm.org/citation.cfm?id=3026877.3026914

[10] Mu Li, David G. Andersen, Jun Woo Park, Alexander J. Smola, Amr Ahmed, Vanja Josifovski, James Long, Eugene J. Shekita, and Bor-Yiing Su. 2014. Scaling Distributed Machine Learning with the Parameter Server. In *Proceedings of the 11th USENIX Conference on Operating Systems Design and Implementation (OSDI'14)*. USENIX Association, Berkeley, CA, USA, 583–598. http://dl.acm.org/citation.cfm?id=2685048.2685095

[11] Mu Li, David G. Andersen, Alexander Smola, and Kai Yu. 2014. Communication Efficient Distributed Machine Learning with the Parameter Server. In *Proceedings of the 27th International Conference on Neural Information Processing Systems (NIPS'14)*. MIT Press, Cambridge, MA, USA, 19–27. http://dl.acm.org/citation.cfm?id=2968826.2968829

[12] Xiaozhou Li, Raghav Sethi, Michael Kaminsky, David G. Andersen, and Michael J. Freedman. 2016. Be Fast, Cheap and in Control with SwitchKV. In *13th USENIX Symposium on Networked Systems Design and Implementation (NSDI 16)*. USENIX Association, Santa Clara, CA, 31–44. https://www.usenix.org/conference/nsdi16/technical-sessions/presentation/li-xiaozhou

[13] Ming Liu, Liang Luo, Jacob Nelson, Luis Ceze, Arvind Krishnamurthy, and Kishore Atreya. 2017. IncBricks: Toward In-Network Computation with an In-Network Cache. *SIGOPS Oper. Syst. Rev.* 51, 2 (April 2017), 795–809. https://doi.org/10.1145/3093315.3037731

[14] Shaohuai Shi, Qiang Wang, Pengfei Xu, and Xiaowen Chu. 2016. Benchmarking State-of-the-Art Deep Learning Software Tools. (2016). arXiv:arXiv:1608.07249

[15] Alexander Smola and Shravan Narayanamurthy. 2010. An Architecture for Parallel Topic Models. *Proc. VLDB Endow.* 3, 1-2 (Sept. 2010), 703–710. https://doi.org/10.14778/1920841.1920931

[16] Ce Zhang and Christopher Ré. 2014. DimmWitted: A Study of Main-memory Statistical Analytics. *Proc. VLDB Endow.* 7, 12 (Aug. 2014), 1283–1294. https://doi.org/10.14778/2732977.2733001