Lasso-ing the Wild!: An Empirical Analysis on Network Topology Effects on Distributed HOGWILD!

Andrei Kozyrev [*1] **Magd Bayoumi** [*1] **Samar Khanna** [*1]

Abstract

As datasets and models continue to grow in size and complexity, a large body of work has focused on parallelizing various aspects of machine learning algorithms, especially Stochastic Gradient Descent (SGD). HOGWILD! (Recht et al., 2011) achieved linear speedups by introducing lock-free updates of a global model vector by each core on a single machine. HOGWILD++ (Zhang et al., 2016) built upon that work and proposed a distributed, asynchronous SGD algorithm that can be run on a cluster of machines. Others have also modified HOGWILD! to adapt it to a multi-machine distributed setting. However, in the distributed setting it remains unclear how network topology and the use of network communication to perform parameter updates affects the performance of parallel SGD algorithms like HOGWILD! and HOGWILD++. We explore the effect of two different network topologies on model training with centralized HOGWILD! and the decentralized HOGWILD++, with star and ring configurations respectively. Similar to results of (Lian et al., 2018), we find that denser network topologies, like the worker-master model, take longer to converge than sparser topologies, like a ring. We also investigate the impact on latency and find that network connections cause a delay in the iterate updates of HOGWILD!, thereby slowing the convergence speed of the algorithm.

1. Introduction

Training large neural network models is a computationally intensive task, especially if it involves a large dataset of images, video or even text. Stochastic Gradient Descent (SGD) is arguably the most widely used optimization algorithm for training deep networks, though its appeal is limited by its "inherently sequential nature," which makes it difficult to parallelize (Recht et al., 2011). Several years ago, the HOGWILD! algorithm was one of the first to introduce fast parallel SGD using asynchronous parallel updates to a single shared parameter memory without locking. Though the shared memory allowed nodes to occasionally overwrite each other's updates, the authors showed that in sparse optimization problems the effect of overwrites is kept at a minimum that still allows for fast convergence. However, (Lian et al., 2018) raised concerns about scalability of Hogwild! because different threads reading and writing to the same shared model memory "results in excessive invalidation of cache lines on writes and an increase in coherence misses." Instead, they proposed the HOGWILD++ algorithm that replaces shared memory with "a set of local model vectors that are shared by a cluster." (Zhang et al., 2016) Synchronization is done via a token-based protocol in which nodes pass their model updates to neighboring nodes. Their results suggest that HOGWILD++ indeed scales better than HOGWILD! while still achieving fast (and in some cases faster) parallel SGD training.

In practice, distributed parallel SGD algorithms, whether synchronous or asynchronous, suffer from network communication bottlenecks. For example, updating the shared parameter memory in a large cluster can cause significant network traffic that may hit the upper limit of the network's throughput. Similarly, latency in network communication can delay parameter updates or lead to unsustainable packet loss. Such scenarios are likely to happen in real-world applications of distributed machine learning, such as when training data is distributed across different geographical locations (sometimes for convenience, other times for legal reasons if we think about multinational financial organizations) or is simply too large and messy to centralize (Verbraeken et al., 2019). In this paper, we focus on investigating the impact of network topology (i.e. how the worker nodes are connected to each other and the shared memory node if applicable) on SGD in the distributed use case, using the HOGWILD! and HOGWILD++ algorithms to distribute and parallelize SGD. In addition, we investigate the impact of these topologies on latency and how it affects the performance of the algorithms.

[*]Equal contribution [1]Department of Computer Science, Cornell University. Correspondence to: Samar Khanna <sak296@cornell.edu>, Andrei Kozyrev <aak85@cornell.edu>, Magd Bayoumi <mb2363@cornell.edu>.

2. Related Work

SGD is a popular learning algorithm for optimising machine learning models, especially with neural networks used in deep learning. Recht et al. demonstrated that to scale SGD to multi-core systems where multiple processes are concurrently running a training job, one doesn't need to use locks on the model weights to achieve convergence (Recht et al., 2011).

HOGWILD! in its original form has a natural sibling in a master-worker distributed network, as explored by Noel et al. (Noel & Osindero, 2014). In this form, a central node acts as the master and is always sending weights to the worker nodes. The worker nodes perform SGD updates and send the difference between the updated local weights and the previous snapshot back to the master. In this scenario, the topology of the network resembles a star; the master node communicates with all workers but each worker does not speak with other workers.

Lian et al. perform a comparison of numerous asynchronous and distributed variants of SGD (including HOGWILD!) and show that for large enough numbers of workers, the communication load with the master node acts as a bottleneck and can negatively impact the convergence rate (Lian et al., 2018). Instead, they propose a *decentralized* version of asynchronous distributed SGD, where the network topology matches that of a clique. Here, given that there is no "master", each worker node randomly picks another node in the ring and averages its weights with those of the other node. Lian et. a. show that this version outperforms asynchronous but *centralized* SGD. However, in their experiments, they only simulate a distributed environment and artifically inject latency into the distributed network.

Furthermore, in HOGWILD!++, Zhang et al. also propose to address the central node bottleneck by considering a ring-like topology, except in their model each node communicates only with its immediate neighbours (Zhang et al., 2016). Their method of averaging the weights involves a "decay" term, so that differences between the local version of the model weights trained with SGD and the synchronised snapshot of the weights do not propagate around the ring to cause divergence. However, in their experiments, the authors do not compare their version with a distributed implementation of HOGWILD!, instead choosing to use the classic asynchronous implementation.

All the above methods pay little to no attention to the effect of real distributed network topologies on convergence. In a real distributed network, nodes can be scattered across the globe and latency can play a role in affecting the convergence of the algorithms. Therefore, we compare a distributed implementation of HOGWILD! with HOGWILD!++ with nodes spread across the continental US. We pick HOGWILD!++ as our distributed decentralized algorithm of choice since it demonstrates strong convergence properties against classical HOGWILD!.

3. Experiments

With our experiments, we (1) evaluate the effect of network topology on the number of epochs to convergence, (2) evaluate the effect of topology on convergence in terms of wall-clock time, and (3) the statistical power of the model after convergence. In addition to topology, we inspect latency in a practical setting to understand the impact of communication over a network as it has been significantly explored with synchronous parallel algorithms but not with asynchronous ones (Dekel et al., 2012; Gimpel et al., 2010; Shalev-Shwartz et al., 2007). We consider the following optimization problem: minimization of Cross-Entropy Loss through a Convolutional Neural Network [1] on the Cifar10 dataset. (Krizhevsky, 2009).

We utilize Google Cloud Platform to create instances as separate nodes. We place nodes within various regions, including US central and Western Europe, using the c2-standard-4 machine type which has 4 CPUs (Intel Cascade Lake) and 16 GB of RAM running the debian OS. We utilize PyTorch's multiprocessing package for training processes (also called threads) sharing one local copy of the model. The PyTorch Distributed package is used to initialize and manage communication between separate nodes in the star and ring topologies over a TCP connection to signal, as well as share, model updates.

We use synchronous SGD and non-distributed HOGWILD! as baselines to see the speed up in a distributed setting and to be able to compare the distributed HOGWILD! with a non-distributed version. We then build a ring implementation of HOGWILD! using Zhang et al.'s work in developing HOGWILD++ (Zhang et al., 2016). We ensure that there are 4 nodes and processes for each of these configurations to ensure a fair comparison in the number of workers, although HOGWILD++ will allow one to have multiple processes at each node. Each node starts with the same model parameters that have been initialized through default PyTorch functions and synchronized prior to training.

4. Results

In this section, we evaluate the performance of the various network topologies and the impact of latency on learning. The main objective of this work is to explore how different asynchronous topologies and node locations impact effi-

[1] The network involved 2 convolutional blocks consisting of a 3x3 convolutional layer with 64 channels, batch normalization and ReLU activation. We then use 2 fully connected layers, one with 256 channels and the other outputting a tensor of length 10

ciency, and our first experiment is to compare the performance of each implementation. Figure 3 and 4 show the objective value on the test set in terms of wall-clock time. Based on these results, we make the following observations:

- HOGWILD++ with ring topology converges much faster than single-machine HOGWILD! over multiple runs for a simple neural network architecture on Cifar10. The suspected reason for this is the dense feature space causing HOGWILD! to have more memory conflicts when accessing the shared model parameters.
- HOGWILD-STAR and HOGWILD++ both contain 4 model replicas and work with two different synchronization strategies (read/overwrite parameter server vs. passing along a ring). Our experiments show that HOGWILD++ is faster than the distributed version of HOGWILD!. This indicates that the Hogwild++ ring performs better than the denser topology of the star while maintaining comparable speed.

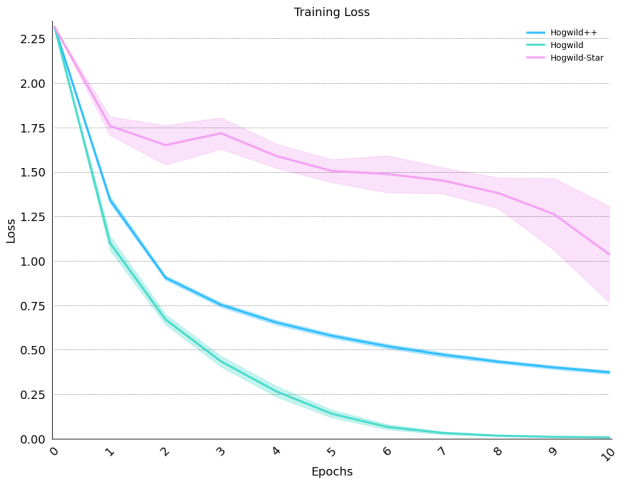

Figure 1. Training loss over epochs for all configurations

Interestingly, we find that the convergence speed between the ring and star is almost the same which contradict theoretical findings that predict converges to be faster on a clique but align with experimental results found in (Luo et al., 2019; Lian et al., 2018; 2017). In terms of wall clock time, (Lian et al., 2018) found that convergence is faster for sparser topologies an effect they attributed to a smaller communication load. We find that the ring does converge faster than the star although difference in wall-clock time is not significant. The convergence is very dependent on the update rate of the algorithm, in terms of how many iterations before a synchronize operation is called.

We find that updating in every iteration takes a time similar to that of a non-distributed HOGWILD! confirming the results from (Lian et al., 2018). We report results with an update rate of every 100 updates (rate at which HOGWILD! communicates with the parameter server and ring passes tokens in HOGWILD++), to reduce communication load differences. This does in fact cause their convergence time to be similar although the star had more communication load per node.

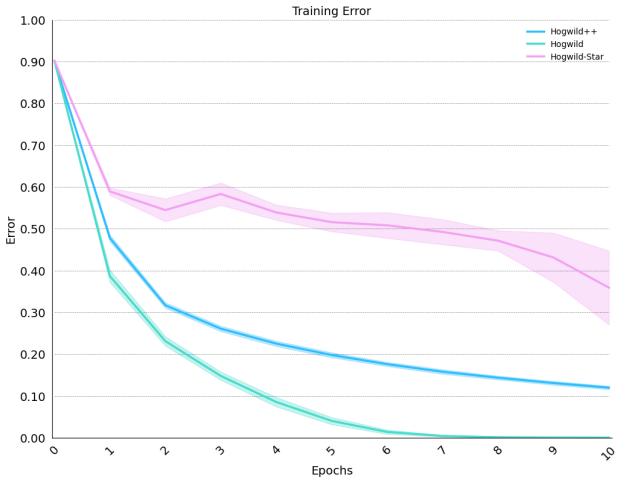

Figure 2. Training error over epochs for all configurations

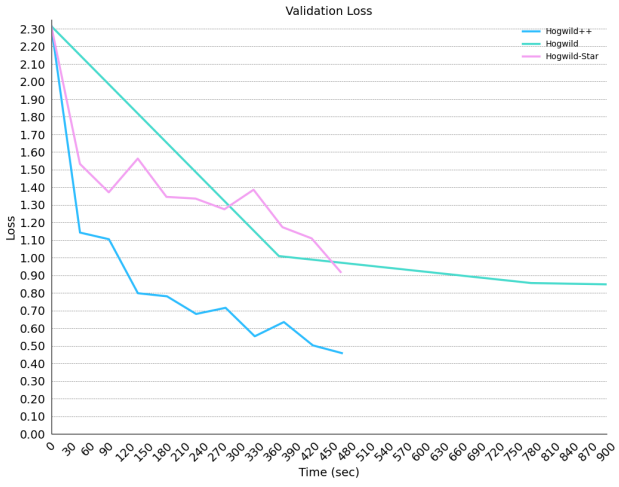

Figure 3. Validation loss over time

We also find that although in theory, the bigger the spectral gap, it should take fewer iterations to converge. However, our experiments do not show a difference in the convergence rate with respect to iterations even when the spectral gaps are dissimilar. Although this is only a comparison of two spectral gaps and one would need more to truly make any empirical statements, this does pose an interesting question of how much communication latency impact the conver-

gence of the model, and whether different topologies have a difference in communication latency.

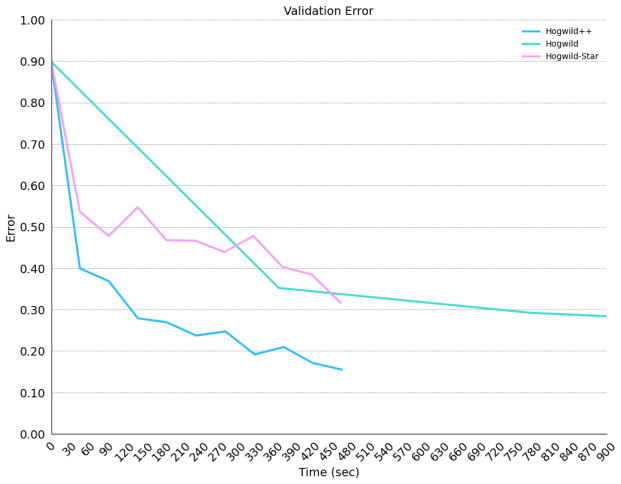

Figure 4. Validation error over time

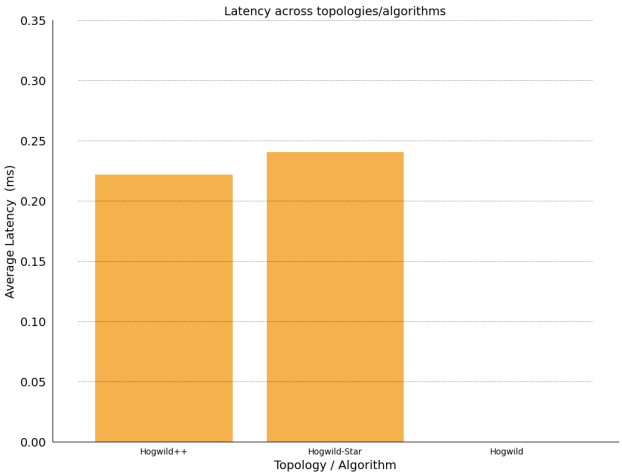

Figure 5. Communication latency over network

We clearly see that even with the delayed updates the star topology has a slightly higher latency although this is not a significant enough difference to cause a difference in the overall communication load which explains the speed of converge being similar.

The second experiment focused on statistical power of each of the algorithms after they had started to converge around 10 epochs. We do note that the traditional HOGWILD! algorithm does seem to perform better in terms of statistical power than HOGWILD-STAR. We hypothesize that this happens due to the communication cost of the distributed structure causing additional parameter update delays. We can see in Figure 6 and 7 that the validation loss consistently decreases for HOGWILD! and has more erratic – although still downward trend – for the distributed algorithms. This helps confirm that the communication delays from the distributed platforms do cause some potential divergence issues (similar updates being compounded from different nodes due to delay) which are less visible in HOGWILD++ due to the decay factor as the updates are passed around the ring, so that when it arrives back at node i, there is little to no amplification.

We conclude from this that distributed platforms are also able to converge and achieve similar statistical power as a shared model on a single node, although the topology does have an impact on its performance with the ring topology surpassing the star within the first epoch.

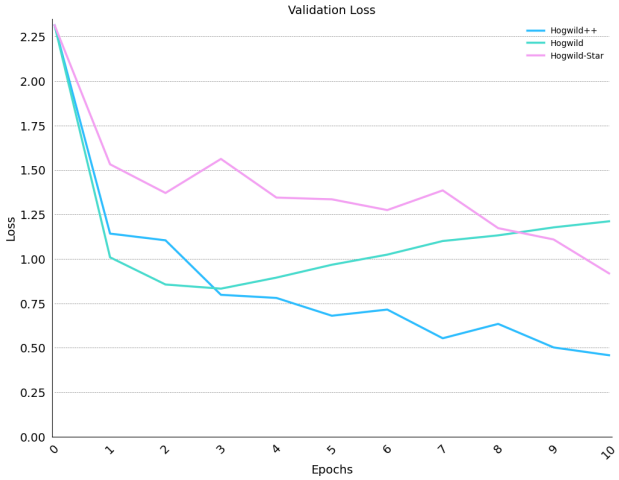

Figure 6. Validation loss over epochs

5. Discussion

We explore the effect of two network topologies, a ring and a star, on the training performance of a commonly used and well-known parallel SGD algorithm HOGWILD! and the promising, decentralized parallel version in HOGWILD++. In our experiments, we start with a baseline, non-distributed implementation of HOGWILD! as suggested in the original paper, and then compare it to a distributed version and to HOGWILD++, both of which we implemented. Our key finding is that there is a fundamental trade-off between the number of connections in a distributed system that runs one of these algorithms and latency. In particular, the topology of a star led to slower convergence even when computing nodes were located in the same geographic region. We find that the ring topology presents significant improvements in speed of model convergence and is less affected by latency,

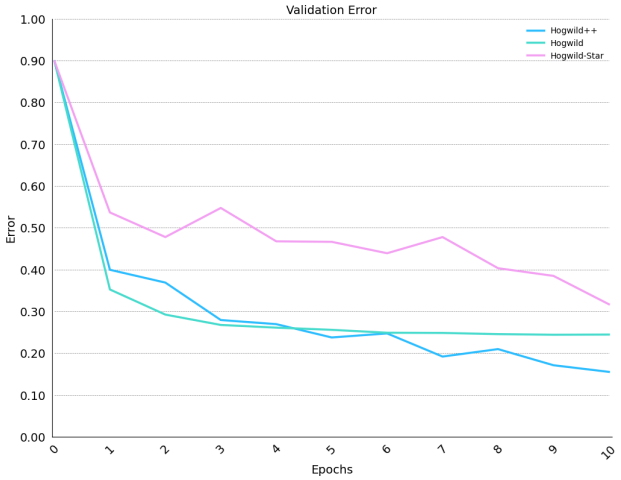

Figure 7. Validation error over epochs

which in turn improves wall-clock time it takes to train the model.

While our work does not provide a theoretical basis for why certain topologies impact these algorithms in the way we see in our experimental results, this can certainly be a fruitful direction for future research. Another opportunity for future work is in developing faster, asynchronous, decentralized algorithms with ring topologies for emerging problem spaces like Federated Learning or other machine learning tasks on edge devices. As machine learning systems become more decentralized, so should the algorithms that make them possible.

References

Dekel, O., Gilad-Bachrach, R., Shamir, O., and Xiao, L. Optimal distributed online prediction using mini-batches. *J. Mach. Learn. Res.*, 13(1):165–202, January 2012. ISSN 1532-4435.

Gimpel, K., Das, D., and Smith, N. A. Distributed asynchronous online learning for natural language processing. In *Proceedings of the Fourteenth Conference on Computational Natural Language Learning*, pp. 213–222, Uppsala, Sweden, July 2010. Association for Computational Linguistics. URL https://www.aclweb.org/anthology/W10-2925.

Krizhevsky, A. Learning multiple layers of features from tiny images. Technical report, 2009.

Lian, X., Zhang, C., Zhang, H., Hsieh, C.-J., Zhang, W., and Liu, J. Can decentralized algorithms outperform centralized algorithms? a case study for decentralized parallel stochastic gradient descent. In Guyon, I., Luxburg, U. V., Bengio, S., Wallach, H., Fergus, R., Vishwanathan, S., and Garnett, R. (eds.), *Advances in Neural Information Processing Systems*, volume 30, pp. 5330–5340. Curran Associates, Inc., 2017. URL https://proceedings.neurips.cc/paper/2017/file/f75526659f31040afeb61cb7133e4e6d-Paper.pdf.

Lian, X., Zhang, W., Zhang, C., and Liu, J. Asynchronous decentralized parallel stochastic gradient descent, 2018.

Luo, Q., Lin, J., Zhuo, Y., and Qian, X. Hop. *Proceedings of the Twenty-Fourth International Conference on Architectural Support for Programming Languages and Operating Systems*, Apr 2019. doi: 10.1145/3297858.3304009. URL http://dx.doi.org/10.1145/3297858.3304009.

Noel, C. and Osindero, S. Dogwild! – distributed hogwild for cpu gpu. 2014.

Recht, B., Re, C., Wright, S., and Niu, F. Hogwild: A lock-free approach to parallelizing stochastic gradient descent. In Shawe-Taylor, J., Zemel, R. S., Bartlett, P. L., Pereira, F., and Weinberger, K. Q. (eds.), *Advances in Neural Information Processing Systems 24*, pp. 693–701. Curran Associates, Inc., 2011. URL http://papers.nips.cc/paper/4390-hogwild-a-lock-free-approach-to-parallelizin
pdf.

Shalev-Shwartz, S., Singer, Y., and Srebro, N. Pegasos: Primal estimated sub-gradient solver for svm. In *Proceedings of the 24th International Conference on Machine Learning*, ICML '07, pp. 807–814, New York, NY, USA, 2007. Association for Computing Machinery. ISBN 9781595937933. doi: 10.1145/1273496.1273598. URL https://doi.org/10.1145/1273496.1273598.

Verbraeken, J., Wolting, M., Katzy, J., Kloppenburg, J., Verbelen, T., and Rellermeyer, J. S. A survey on distributed machine learning, 2019.

Zhang, H., Hsieh, C.-J., and Akella, V. Hogwild++: A new mechanism for decentralized asynchronous stochastic gradient descent. *2016 IEEE 16th International Conference on Data Mining (ICDM)*, pp. 629–638, 2016.