Centralized vs Decentralized Stochastic Optimization Algorithms

Arijit Pramanik, Deepak Srinath and Hemant Chinchore - Group 5

1 Introduction

Federated learning entails training statistical models directly over a large number of remote devices using their local data. These devices individually collect training samples and compute gradients locally. Due to the growing computational power of consumer mobile devices and IoT devices coupled with privacy concerns over transmitting raw training data, it is becoming increasingly attractive to store data locally and push network computation to the edge. In contrast, classic ML training approaches require centralizing the training process to a local machine or distributing the dataset over dedicated training nodes within large datacenters. The downside of this architecture is that the data collected by local devices/sensors have to be sent back to the central cloud servers for processing wherein the trained model is subsequently communicated back to the devices for real-time on-device inference.

Federated learning brings in the ability to take advantage of distributed edge resources like consumer devices to train machine learning models remotely. Edge devices synchronize with the current model from the server, improve it by learning from local data on device and summarize "learning" as a small focused update. Only such model updates, e.g. on-device computed gradients are sent to the cloud, using encrypted communication, where they are immediately averaged with other device updates to improve the global shared model. Other approaches aim to simultaneously learn distinct device-specific models via multi-task learning frameworks. Even though federated learning has found success in numerous applications such as self-driving cars, smart manufacturing, digital health and tasks like next-word prediction, it comes with its own set of challenges. (1) The most important concern in a federated setting is the privacy of localized data. Though raw data is never communicated, sensitive information may still be revealed to a third party/central server. Organizations or institutions such as hospitals can also be viewed as *devices* in the context of federated learning. However, hospitals operate under strict privacy practices, and hence may be unwilling to trust a third party even with only the model updates. Approaches used to enhance the privacy of federated learning such as multiparty computation or differential privacy often cause reduced model performance and system efficiency. (2) Since thousands of devices are involved in every training step, the communication costs can be at par with computation complexity. (3) The devices involved in the training step are heterogeneous in nature and differ in terms of storage, computational and communication capabilities due to variability in hardware, network connectivity and power. The associated unreliability of network and these devices might result in only a handful of them *actively* participating in an ongoing training iteration during which any of them may arbitrarily drop out. Hence, federated learning should also be robust to device failures and stragglers. (4) The data collected by these devices can be statistically heterogeneous and may violate frequently used independent and identically distributed (I.I.D.) assumptions in distributed optimization which can further complicate the process of analysis and evaluation of the trained model.

Communication is a key bottleneck in federated networks and communication in the network can be slower than local computation by many orders of magnitude owing to $O(n)$ communication overhead at each node for n nodes. Most machine learning frameworks are built on top of centralized approaches which use a star network topology like parameter servers and gradient consensus, wherein a designated node gathers gradients and scatters the mean gradient values. These approaches introduce a bottleneck on the central node which is exacerbated in a federated setting. Such bottlenecks can be circumvented by taking decentralized approaches with communication complexity of $O(deg(network))$ promising asymptotic linear speedup since a star topology may not always be available. Even in such decentralized approaches, communication may involve propagation of locally computed updates across all nodes over several rounds of communication (e.g. AllReduce). Several approaches limit propagation of such information to a set of immediate neighbours. The amount of data transmitted can also be reduced by gradient compression techniques like sparsification and quantization. Hence we aim to study and analyze the performance of decentralized algorithms over centralized counterparts in a data-parallel federated setting on various network topologies (e.g. ring, torus) with different latency and bandwidth guarantees. We also compare the use of biased and unbiased compression operators along with communication-efficient methods that can reduce the number of rounds of communication.

A special variant of stochastic gradient descent called parallel stochastic gradient descent (PSGD) is normally used for distributed machine learning training and several variants of this algorithm such as centralized (synchronous SGD with parameter server) and D-PSGD (decentralized) - synchronous and asynchronous versions have been theoretically compared. In a distributed setting, it is important to find the average gradient vector over n local gradient vectors from (say) n nodes and usually gossip algorithms are leveraged to find such an average vector. Several variants of gossip algorithms scale very well while dealing with compressed communication. To this end, we also study the state-of-the-art communication-efficient techniques implemented on top of D-PSGD.

2 Background

Stochastic Gradient Descent: *SGD* is a standard algorithm used for machine learning problems. It is an inherently serial algorithm that does not take distributed setting into account. *Mini-batch SGD* is the natural parallelization of *SGD* in a centralized setting.

Decentralized Communication: The network topology is often modelled as a graph. Centralized topologies corresponding to a *star* graph pose a significant bottleneck on the central node in terms of communication latency, bandwidth and fault tolerance. The master node in a centralized setting receives (sends) messages from (to) all workers in each round, $\Theta(n)$ in total where n is the number of nodes. In decentralized topologies, e.g. ring or torus, the maximal degree of the network is often constant or a slowly growing function in n.

Decentralized Optimization: Each training iteration in case of centralized optimization starts only when all nodes have the same global model, e.g. Ring All-Reduce, whereas in decentralized optimization, since each node updates its model based on updates only from neighbors, it is possible that each node has a slightly different model. Recent work [11] shows that decentralized algorithms (also termed *gossip* algorithms) can provide similar convergence guarantees as their centralized counterparts.

Communication Compression: In distributed ML training, model updates (or gradient vectors) have to be exchanged between the worker nodes. To reduce the amount of data to be sent over the network, several *gradient compression* strategies using *quantization* [3] and *sparsification* ([2]; [19]) are used in practice.

Federated Learning: The canonical federated learning problem involves learning a *single, global* statistical model from data stored on tens to potentially millions of remote devices. The goal is typically to minimize the following objective function: $min_w F(w)$ where $F(w) := \sum_{k=1}^{m} p_k F_k(w)$, where m is the total number of devices, $p_k \geq 0, \sum_k p_k = 1$ and F_k is the local objective function for the kth device.

3 Centralized vs Decentralized

Throughout the early Big Data era, most of the system design was driven by performance, accuracy, resource usage and usability. But, federated learning introduces a new requirement of privacy. Privacy and security have always been considered to be in friction with performance and usability. Hence, a federated setting might need to compromise on performance in order to achieve privacy.

In order to understand how much of a compromise needs to be done in terms of performance and accuracy, we ran an experiment to compare centralized and decentralized algorithms on CloudLab in an environment composed of 8 Ubuntu 18.04 nodes. *CIFAR-10* dataset was used to train a *ResNet-20* model. The centralized algorithm made use of a parameter server while decentralized approach entails use of the D-PSGD algorithm. Further, Ring AllReduce was used as it uses a centralized synchronization model in a decentralized topology, wherein all nodes need not be directly connected.

Decentralized Stochastic Gradient Descent (D-PSGD)[11] is an algorithm where each worker owns a local copy of the model. For each iteration, each worker computes the stochastic gradient and averages its local model with its neighbors. The centralized algorithm utilizes a parameter server that receives the gradients from all the other nodes, calculates the average gradient and broadcasts it back to the workers. In order to achieve the functionality of a parameter server, we make use of gather and `scatter` approach. One master node is used to gather all the gradients and again scatter them to all the worker nodes. Ring-AllReduce is specifically used as the gradients for each iteration is passed onto all the devices in an network bandwidth-optimized way. While in D-PSGD, a node just distributes its gradient to its immediate neighbors. Fig 1 shows how gradients are distributed in each of the approaches mentioned above. In the case of a parameter server, the gradients are all pushed towards the central node. Hence, we expect the central node to have more traffic compared to other nodes. In case of D-PSGD each node sends the gradients to its neighbor as well as receives gradients from its neighbors and hence each node has same amount of traffic going through it. But, it might be twice the size of a gradient. Finally, Ring AllReduce sends one gradient to a neighbor and receives a gradient from another neighbor. Hence, each node will have similar network traffic and the amount of data sent (or received) will be

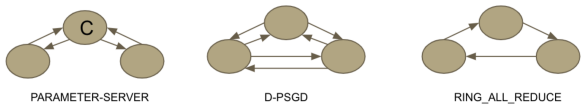

Figure 1: Gradient distribution in parameter-server, D-PSGD and Ring AllReduce

equal to the size of a gradient vector.

3.0.1 Network Usage

The network usage was the first parameter that was measured as this is crucial in a federated learning setting. The network usage of only those nodes that might be a bottleneck in each algorithm was measured. In the case of centralized parameter server, the master node where the gradients are accumulated would be the potential bottleneck as all the workers send gradients and receive updated gradients from it. In the case of D-PSGD, each node would have similar network usage. Similarly, Ring AllReduce would also have similar network usage across all its nodes. One of the main observations from the description of these algorithms would be that the network usage would be twice for each node in D-PSGD compared to that of AllReduce in `DistributedDataParallel` setting. But, this communication would be constant and is independent of the number of nodes. The parameter server in centralized algorithm can be considered to be a function of the number of nodes and as the number of nodes increase, more number of gradients would be passed to the parameter server.

Figure 2 shows the network usage as experimented on the Cloudlab environment. Some of the major takeaways from the graph are that the Ring AllReduce has the least network usage. As expected, D-PSGD has twice the number of gradients transferred across when compared to the Ring AllReduce. Parameter server has around 15MB of data transfer, which can be attributed to 7 worker nodes sending and receiving gradients. But, this amount of data might increase if the number of nodes participating in training increases. Similar reasoning can be extended to D-PSGD as well as Ring AllReduce. The other observation is that in each of the algorithms, the network usage hits 0. This shows that there is no complete overlap of communication and computation. From Fig 2 we can also see that the time taken to run 20 epochs is more in the case of parameter server while D-PSGD and Ring AllReduce have similar completion times.

3.0.2 Statistical efficiency

The next concern when comparing centralized and decentralized algorithms is accuracy. As for each iteration,

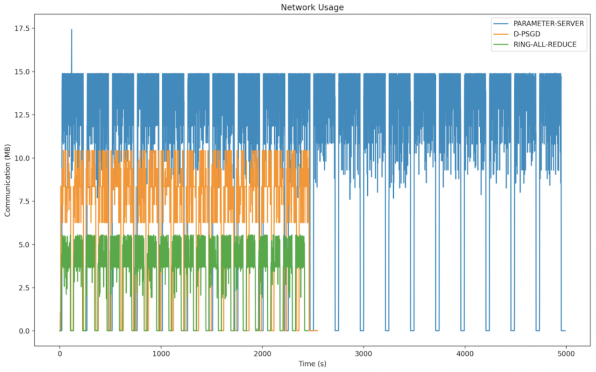

Figure 2: Maximum network usage in parameter-server, D-PSGD and Ring AllReduce

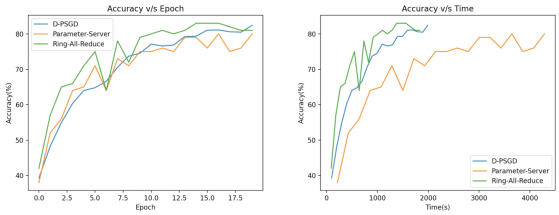

Figure 3: Accuracy vs Epoch and Accuracy vs Time

the amount of gradient data exchanged is very limited for D-PSGD when compared to AllReduce and parameter server. Looking at Fig (3), we see that for 20 epoch, all the algorithms have very similar accuracies. Further, the time taken for each epoch is significantly lower when compared to the parameter server approach.

From this simple experiment, we conclude that the decentralized algorithm achieves accuracy comparable to the centralized approach and makes efficient utilization of the resources. These results reinforce our assumption that decentralized algorithms can achieve privacy, as well as perform at par with centralized algorithms. Hence, decentralized algorithms can be used in federated settings that do not want to rely on a centralized server.

3.1 Mini-batch SGD variants

Parallel Stochastic Gradient Descent (PSGD) [25] divides a dataset into small batches across machines where gradients are computed for the entire mini-batch and synchronized across machines. After averaging gradients and batch norm synchronization, model parameters are updated accordingly. For centralized (C-PSGD), gradients are synchronized across all machines (a fully-connected or master-worker parameter server topology). Elastic Averaging (EA-SGD) [23] maintains parameters local to workers and global center variables, focusing on reducing communication between master and workers.

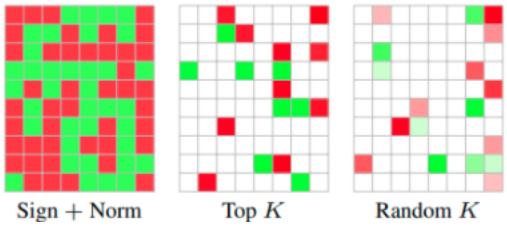

Figure 4: Different gradient compression schemes [22]

The elasticity parameter ensures that workers don't fall into local optima farther away from center variables.

Decentralized PSGD (D-PSGD) synchronizes gradients only among neighbors leading to idle time in each iteration for some workers while waiting for others to compute and send gradients. Asynchronous D-PSGD (AD-PSGD) [12] eliminates the need to wait at individual worker nodes, by maintaining and updating each worker's local model and synchronizing with local models of neighbouring workers. This might lead to the use of stale models from slow workers but speeds up training in the face of stragglers and slow networks as evident in today's datacenters. Inspired from Quantized SGD (QSGD) [1], we show various compression schemes used in conjunction with SGD in Fig 4. We study Difference Compression D-PSGD (DCD-PSGD) and Extrapolation Compression D-PSGD (ECD-PSGD) [21] which quantize the *difference* and *extrapolation* between the last two local models on each worker respectively. DCD-PSGD offers a slightly better convergence rate when the data variation among nodes is very large, while ECD-PSGD is more robust to aggressive quantization. ChocoSGD [7] supports all biased and unbiased compression operators shown in Fig 4 and is a provably-converging gossip algorithm for the distributed average consensus problem, critical for the federated setting.

Protocols described above entail each node sending updates to all of its neighbours ("push") and receiving an update from each such neighbor ("pull"), and hence called "push-pull" algorithms. Such topologies are typically undirected graphs with a symmetric, row stochastic mixing matrix (row entries summing to 1) where each non-zero entry signifies that nodes with the corresponding row index and column index are connected. Stochastic Gradient Push (SGP) [4] works for undirected, time-varying graphs with one-directional communication wherein each node broadcasts updates and PUSHSUM weights to its out-neighbors for updating the model. Its variant, Synchronous Overlap SGP allows for computation-communication overlap with performance at par with AD-PSGD.

Type	CPUs	L1	L2	LLC
c220g1	40	32 KB	256 KB	25600 KB

Table 1: Cloudlab experiment machine configuration

4 Evaluation

4.1 Experimental setup

We experiment with some simple topologies as shown in Fig 5 6, 7 and 8 to simulate a network of federated devices. We use the *epsilon* dataset [18] to evaluate a simple ML classification task of predicting labels. The dataset consists of 40,000 examples each comprising 2000 features.

We use a Cloudlab machine as seen in Table 1 to evaluate a simple ring topology of 40 devices each pinned to a single CPU core. We observe how the training error and loss function evolve over time with number of epochs and transmitted bits. We measure communication in terms of bytes transmitted, by accounting for the physical size of the `numpy` matrices that are sent/fetched. We use the cross entropy loss function from logistic regression to measure loss and measure training error as the number of incorrect predictions. We use a variety of learning techniques and tune such hyper-parameters following related papers and simple grid search. We show results primarily for training error as we obtain similar results for training loss. We also plot training error and loss vs time but this is not an accurate measure of training time since there is no physical communication among devices which can be affected by heterogeneous network bandwidth and can be a bottleneck compared to the current simulation via a matrix multiplication. We expect similar results on any physical cluster as well. Please refer to our repository for more details.

4.2 Comparing popular SGD variants

We compare PSGD on a fully-connected topology (C-PSGD), synchronous and asynchronous versions of EASGD with momentum (EAMSGD-sync and EAMSGD-async), vanilla D-PSGD, momentum variant of SGP and AD-PSGD. For EAMSGD-async, we synchronize local worker gradients with the central parameter server every 4 mini-batch iterations. From Fig 9 and 21, we observe centralized algorithms communicate much more bytes than decentralized ones for very similar accuracies (i.e. greater X-values for the same Y-value). SGP communicates both gradients and PUSHSUM weights leading to more bytes transmitted. EAMSGD-async also achieves similar accuracy for lesser bytes transmitted than EAMSGD-sync and C-PSGD. The final accuracy obtained by these algorithms is presented in Table 2.

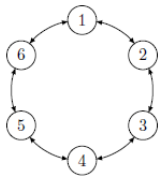

Figure 5: Ring topology

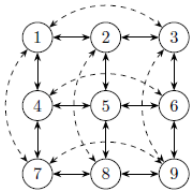

Figure 6: Torus topology

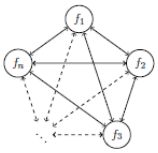

Figure 7: Fully-connected topology

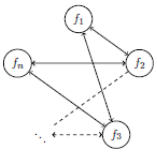

Figure 8: General topology

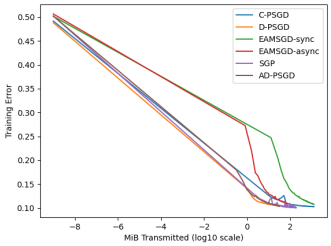

Figure 9: Training error vs bytes communicated for different variants of SGD

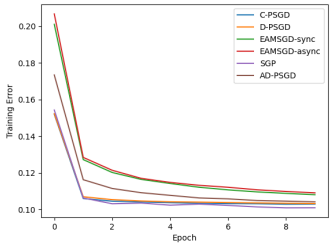

Figure 10: Training error vs epoch (1000 iterations) for different variants of SGD

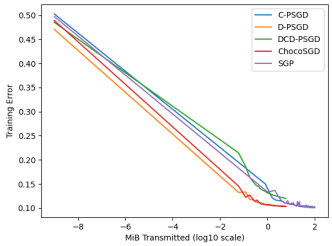

Figure 11: Training error vs bytes communicated for 4-bit gradient quantization

Algorithm	Plain	4-bit qnt.	Top 1%	Random 1%
C-PSGD	89.73	89.70	87.61	86.39
EAMSGD (sync)	89.24	-	-	-
EAMSGD (async)	89.12	-	-	-
D-PSGD	89.69	89.64	87.59	86.23
SGP	89.86	89.71	89.54	88.96
AD-PSGD	89.60	-	51.02	-
ChocoSGD	-	89.68	89.65	89.65
DCD-PSGD	-	87.99	-	54.21
ECD-PSGD	-	49.80	-	50.19

Table 2: Accuracy(%) for SGD variants across different compression schemes (*qnt.* implies quantization)

Overall training error and loss decreases with time as shown in Fig 23 and 24. AD-PSGD completes training in almost same time as D-PSGD since our simulation doesn't comprise any stragglers or communication over network. C-PSGD and EAMSGD take more time with synchronous version taking longer than asynchronous version, as expected. Ideally decentralized algorithms take lesser time to complete 10 epochs, but attain higher training error and loss compared to centralized counterparts, which synchronize model across all devices at every iteration as seen in Fig 10 and 22, leading to a trade-off for statistical efficiency.

4.3 Compression schemes

We demonstrate performance for quantizing 64 bit gradients to 8 bits. We obtain order of magnitude decrease in bytes transferred ($10^{1.5}$ MiB compared to 10^2 MiB on average in previous plots). ChocoSGD and D-PSGD achieve similar accuracies with much lesser data exchange in Fig 11. However, DCD-PSGD converges to a suboptimal value leading to higher training error and loss compared to others in Fig 12. ECD-PSGD diverges under this scheme and hence isn't plotted. Fig 25 shows variation in training error with time. Similar results obtained with 8-bit quantized gradients are presented in the Appendix. Now, we randomly select 1% of the gradients (20 out of 2000) which reduces communication volume even further ($10^{0.5}$ MiB on average). As seen in Fig 13, ChocoSGD outperforms others and uses

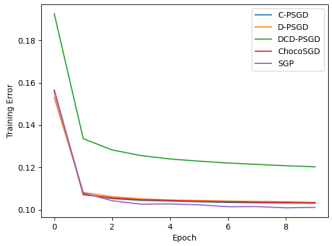

Figure 12: Training error vs epoch (1000 iterations) for 4-bit gradient quantization

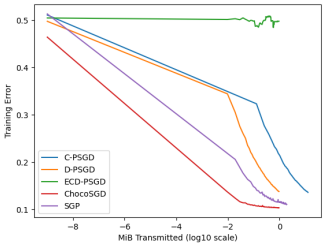

Figure 13: Training error vs bytes communicated for 1% randomly sampled gradients

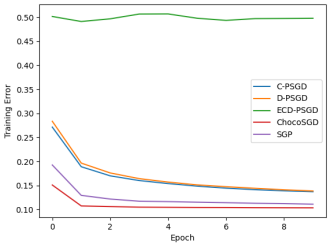

Figure 14: Training error vs epoch (1000 iterations) for 1% randomly sampled gradients

much lesser bytes for lower accuracies in the beginning, though total bytes communicated is close to other decentralized counterparts. DCD-PSGD diverges and isn't plotted while ECD-PSGD fails to converge in this case. SGP attains next lowest training error after ChocoSGD in 14. Training time varies as shown in Fig 26. Compression schemes have been added on top of SGP and we provide no theoretical convergence guarantees. Table 2 shows accuracies achieved with these compression operators.

4.4 Effect of different topologies

We now show the performance of D-PSGD on ring, torus, fully-connected, completely disconnected and a partially connected topology (Erdos-Renyi graph with binomial distribution of vertex degrees implying it is neither dense nor sparse) for 40 devices. From 15, we see that the total number of bytes communicated is least for ring and greatest for fully-connected topology, and varies with the number of neighbours as expected. Training time in Fig 32 remains the same since our simulation with a central weights store is topology-agnostic. From Fig 16, we see that training error and loss are very similar though marginally lesser for ring and marginally greater for centralized topology in Table 3 since the former does gradient synchronization across only 2 neighbors while the latter does so for all devices in the topology.

4.5 Speedup across devices

The classification task involves a simple Neural Network comprising one fully-connected hidden layer, that doesn't pose a *computation bottleneck*. We anticipate a *communication bottleneck* which should slowly increase with the number of devices since there is little benefit and significant overhead of distributing gradient computation over multiple devices. Communication entails a matrix multiplication between the topology mixing matrix and weights matrix consisting of model weights for

Topology vs devices	9	16	25	36
Ring	89.79	89.77	89.75	89.69
Torus	89.82	89.80	89.76	89.72
fully connected	89.77	89.78	89.77	89.75
dis-connected	89.67	89.56	89.43	89.29
partially connected	89.79	89.77	89.79	89.76

Table 3: Accuracy(%) for D-PSGD across different topologies and number of devices

all federated devices. More devices imply multiplication of larger matrices which agrees with more communication overhead with more physical devices.

We demonstrate the performance of D-PSGD on ring topology as we increase the number of devices in 18. Training time is least for 9 devices but increases and remains fairly constant for 16 to 36 devices. We notice that data transferred to attain a given accuracy remains almost same in Fig 17 as the number of neighbours remains unchanged (2 for this ring topology). However, training error and loss decrease marginally with more devices in Fig 31.

4.6 Simulation on physical cluster

In our approach, weights for each device are stored in shared memory which each device (CPU thread) can access to get the weights for its selected subset of neighbors. This bypasses actual communication that would happen over remote network and is agnostic to topology and hence doesn't prove to be a strict bottleneck as could happen in real scenarios. A *container approach* restricts the amount of memory and CPU available to each simulated device. However, the devices don't communicate according to topology and though devices with

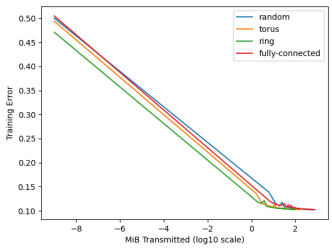

Figure 15: Training error vs bytes communicated for different topologies for D-PSGD

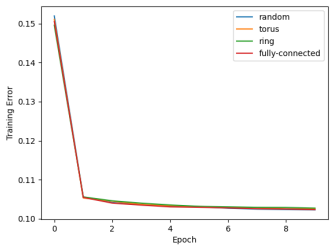

Figure 16: Training error vs epoch (1000 iterations) for different topologies for D-PSGD

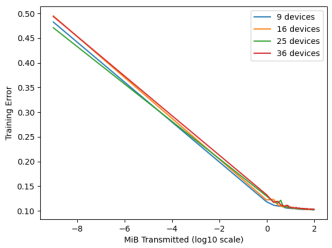

Figure 17: Training error vs bytes communicated for different number of devices for D-PSGD in ring topology

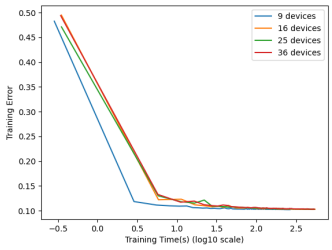

Figure 18: Training error vs time for different number of devices

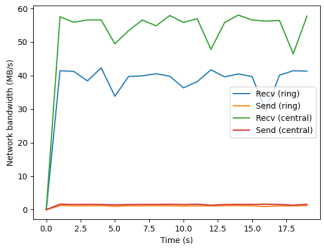

Figure 19: Network usage on remote worker for D-PSGD

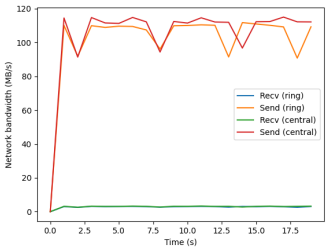

Figure 20: Network usage on GCS machine for D-PSGD

more neighbors fetch more data from shared memory, there is no significant difference since the matrix storing weights for all devices can be easily cached (40 devices $\times$ 2000-dim 8-byte weights = 640 KB) partially within each core's L2 cache and rest in L3 cache.

Hence, we used a simulation of 40 devices spread across 4 `c220g1` machines using a Ray cluster where each device talks to the Ray Global Control Store (GCS) on one of the machines. Using `dstat` in Fig 19, we see a significant difference for ring vs centralized topology since each worker fetches weights for each of the 39 neighbors of each device on remote worker for the centralized topology, whereas in ring topology, weights of only 2 device neighbors are fetched. Similar network receive bandwidth values on remote worker machines are observed for other decentralized topologies. The network send bandwidth of GCS in Fig 20 remains the same across topologies (centralized or decentralized) perhaps due to Ray optimizations. Since our gradient computation is not compute-intensive, scaling out across multiple machines leads to *increase* in training completion time.

5 Related Work

Mini batch SGD is the natural parallelization of SGD, though recently it has faced generalization concerns [13]. Here, we make a background survey of prior work on different aspects of federated learning with our primary focus on communication trade-offs, mainly in decentralized approaches. Li et al [8] discuss several challenges and how to address them. Communication bottlenecks are dealt with using approaches such as local update of model or using compression techniques for exchanging data. Further, the authors discuss about how the privacy issues are dealt in normal machine learning and federated learning settings. Finally, in order to deal with heterogeneous devices, several techniques such as asynchronous communication and active sampling are used. The authors finally point out some future directions to improve federated learning.

Communication Efficiency: For increased flexibility of communication vs computation, a variable number of local updates can be applied to each device in parallel at each communication round. e.g. FedAvg [14] is a method based on averaging local stochastic gradient descent (SGD) updates for the primal problem. In decentralized setting, much work has been done around optimizing gossip protocols [7] to suit federated learning. This introduced a state-of-the-art decentralized stochastic optimization technique and has experimentally demonstrated the performance gain from the newly introduced consensus and SGD algorithms. Lian et al [11] have studied and compared decentralized-PSGD with centralized-PSGD. The authors have empir-

ically proved that the distributed algorithm works at the same level as the centralized approach. But, thanks to addressing the communication bottleneck in decentralized approach, the authors have shown that decentralized approach is preferable for our use case. Compression techniques include random sparsification [19] to randomly mask input vectors and only preserve a constant number of coordinates.

Systems Heterogeneity: To mitigate stragglers prevalent in a federated setting and achieve speedup, asynchronous versions of centralized SGD [10, 6] have been studied with bounded staleness of gradients. Simply ignoring device failures [5] can introduce bias. Coded computation can tolerate device failures by introducing algorithmic redundancy with recovery of the true gradients using replication of data blocks and gradient recomputation [20]. Nishio and Yonetani [16] explore novel device sampling policies based on system resources, with the goal being for the server to aggregate as many device updates as possible within a pre-defined time window.

Statistical Heterogeneity: Multi-task learning frameworks like MOCHA [17] have been designed for the federated setting. Zhao et al [24] explore transfer learning for personalization by running FedAvg after training a global model centrally on some shared proxy data. To understand the performance of FedAvg in statistically heterogeneous setting, FedProx [9] has recently been proposed. FedProx makes a small modification to the FedAvg method to help ensure convergence. Approaches that address fairness other than accuracy like Agnostic Federated Learning [15] optimize the centralized model for a target distribution from a mixture of client distributions.

6 Future work

Our project initially focused on comparing centralized and decentralized algorithms. Various decentralized algorithms were run on different topologies. The experiments were conducted on standard Cloudlab nodes where all the worker nodes and network interconnects were homogeneous in terms of resources. It would be interesting to run the same experiments on heterogeneous machines.

One way to achieve heterogeneity is to have workers geographically distributed with some subset of machines being relatively closer which would impact network bandwidth. The experiments were run on CPUs and the same experiments can be run on machines with GPUs, TPUs or any combination of them. In a federated setting, there is a good chance that some workers would be inactive and will not be part of the training. With this assumption in mind, each of these algorithms can be compared based on robustness to node failures. The relationship between robustness and accuracy can also be studied keeping these failures in mind. Decentralized algorithms can be further optimized through asynchronous computation-communication overlap and different compression techniques. These results would give us a better perspective on decentralized algorithms in real-world federated settings. We propose an approach to deal with such federated settings by making use of differential privacy techniques to enhance privacy.

References

[1] D. Alistarh, D. Grubic, J. Li, R. Tomioka, and M. Vojnovic. Qsgd: Communication-efficient sgd via gradient quantization and encoding. *Advances in Neural Information Processing Systems*, 30:1709–1720, 2017.

[2] D. Alistarh, T. Hoefler, M. Johansson, S. Khirirat, N. Konstantinov, and C. Renggli. The convergence of sparsified gradient methods, 2018.

[3] D. Alistarh, J. Li, R. Tomioka, and M. Vojnovic. QSGD: randomized quantization for communication-optimal stochastic gradient descent. *CoRR*, abs/1610.02132, 2016.

[4] M. Assran, N. Loizou, N. Ballas, and M. Rabbat. Stochastic gradient push for distributed deep learning. In *International Conference on Machine Learning*, pages 344–353. PMLR, 2019.

[5] K. Bonawitz, H. Eichner, W. Grieskamp, D. Huba, A. Ingerman, V. Ivanov, C. Kiddon, J. Konečnỳ, S. Mazzocchi, H. B. McMahan, et al. Towards federated learning at scale: System design. *arXiv preprint arXiv:1902.01046*, 2019.

[6] S. Dutta, J. Wang, and G. Joshi. Slow and stale gradients can win the race. *arXiv preprint arXiv:2003.10579*, 2020.

[7] A. Koloskova, S. U. Stich, and M. Jaggi. Decentralized stochastic optimization and gossip algorithms with compressed communication, 2019.

[8] T. Li, A. K. Sahu, A. Talwalkar, and V. Smith. Federated learning: Challenges, methods, and future directions. *IEEE Signal Processing Magazine*, 37(3):50–60, May 2020.

[9] T. Li, A. K. Sahu, M. Zaheer, M. Sanjabi, A. Talwalkar, and V. Smith. Federated optimization in heterogeneous networks. *arXiv preprint arXiv:1812.06127*, 2018.

[10] X. Lian, Y. Huang, Y. Li, and J. Liu. Asynchronous parallel stochastic gradient for nonconvex optimization. In *Advances in Neural Information Processing Systems*, pages 2737–2745, 2015.

[11] X. Lian, C. Zhang, H. Zhang, C.-J. Hsieh, W. Zhang, and J. Liu. Can decentralized algorithms outperform centralized algorithms? a case study for decentralized parallel stochastic gradient descent, 2017.

[12] X. Lian, W. Zhang, C. Zhang, and J. Liu. Asynchronous decentralized parallel stochastic gradient descent. In *International Conference on Machine Learning*, pages 3043–3052. PMLR, 2018.

[13] T. Lin, S. U. Stich, K. K. Patel, and M. Jaggi. Don't use large mini-batches, use local sgd. *arXiv preprint arXiv:1808.07217*, 2018.

[14] B. McMahan, E. Moore, D. Ramage, S. Hampson, and B. A. y Arcas. Communication-efficient learning of deep networks from decentralized data. In *Artificial Intelligence and Statistics*, pages 1273–1282. PMLR, 2017.

[15] M. Mohri, G. Sivek, and A. T. Suresh. Agnostic federated learning. *arXiv preprint arXiv:1902.00146*, 2019.

[16] T. Nishio and R. Yonetani. Client selection for federated learning with heterogeneous resources in mobile edge. In *ICC 2019-2019 IEEE International Conference on Communications (ICC)*, pages 1–7. IEEE, 2019.

[17] V. Smith, C.-K. Chiang, M. Sanjabi, and A. S. Talwalkar. Federated multi-task learning. In *Advances in Neural Information Processing Systems*, pages 4424–4434, 2017.

[18] S. Sonnenburg, V. Franc, E. Yom-Tov, and M. Sebag. Pascal large scale learning challenge, 2008.

[19] S. U. Stich, J.-B. Cordonnier, and M. Jaggi. Sparsified sgd with memory. In *Advances in Neural Information Processing Systems*, pages 4447–4458, 2018.

[20] R. Tandon, Q. Lei, A. G. Dimakis, and N. Karampatziakis. Gradient coding: Avoiding stragglers in distributed learning. In *International Conference on Machine Learning*, pages 3368–3376, 2017.

[21] H. Tang, S. Gan, C. Zhang, T. Zhang, and J. Liu. Communication compression for decentralized training. In *Advances in Neural Information Processing Systems*, pages 7652–7662, 2018.

[22] T. Vogels, S. P. Karimireddy, and M. Jaggi. Powersgd: Practical low-rank gradient compression for distributed optimization. In *Advances in Neural Information Processing Systems*, pages 14259–14268, 2019.

[23] S. Zhang, A. E. Choromanska, and Y. LeCun. Deep learning with elastic averaging sgd. In *Advances in neural information processing systems*, pages 685–693, 2015.

[24] Y. Zhao, M. Li, L. Lai, N. Suda, D. Civin, and V. Chandra. Federated learning with non-iid data. *arXiv preprint arXiv:1806.00582*, 2018.

[25] M. Zinkevich, M. Weimer, L. Li, and A. J. Smola. Parallelized stochastic gradient descent. In *Advances in neural information processing systems*, pages 2595–2603, 2010.

7 Appendix

Please refer to the next page.

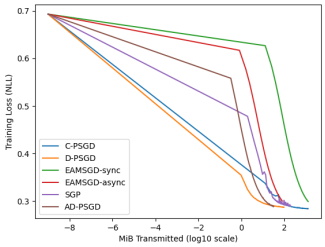

Figure 21: Training loss vs bytes communicated for different variants of SGD

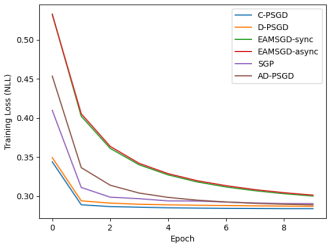

Figure 22: Training loss vs epoch (1000 iterations) for different variants of SGD

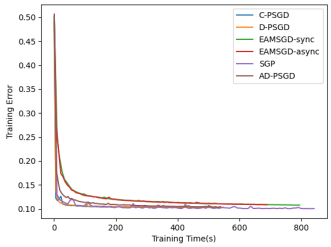

Figure 23: Training error vs time (in seconds) for different variants of SGD

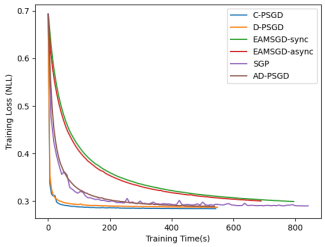

Figure 24: Training loss vs time (in seconds) for different variants of SGD

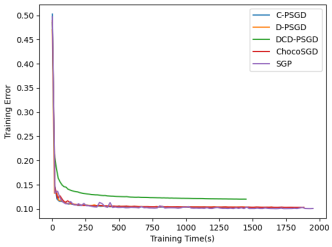

Figure 25: Training error vs time (in seconds) for 4-bit quantized gradients

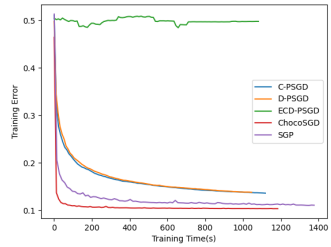

Figure 26: Training error vs time (in secs) for 1% randomly sampled gradients

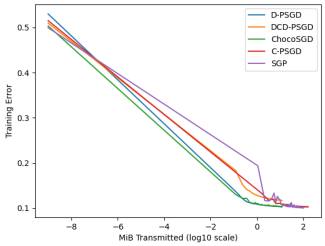

Figure 27: Training error vs bytes communicated for 8-bit quantized gradients

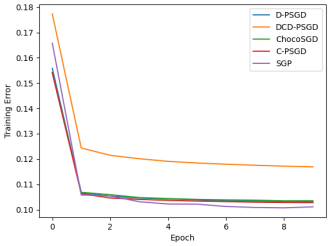

Figure 28: Training error vs epoch (1000 iterations) for 8-bit quantized gradients

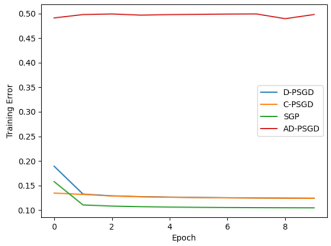

Figure 29: Training error vs epoch (1000 iterations) for top 1% gradients

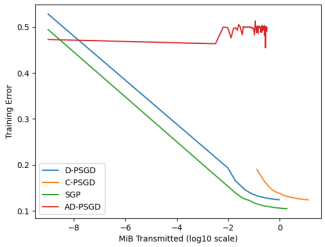

Figure 30: Training error vs bytes communicated for top 1% of gradients

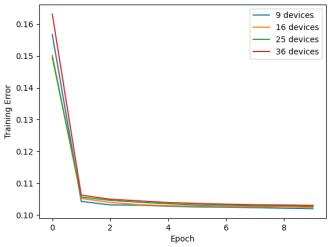

Figure 31: Training error vs epoch for different number of devices for D-PSGD ring topology

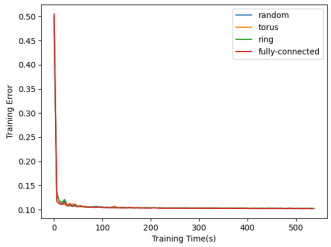

Figure 32: Training error vs time (in seconds) for different topologies for D-PSGD