FiL∅: Dynamic Topologies for Federated Learning

spiros thanasoulas (st19@illinois.edu)

May 8, 2021

Abstract

As the amount of data and participating nodes increases, decentralized and asynchronous versions of machine learning algorithms are becoming more prominent. Overlay networks that exploit efficient topologies have been shown recently to dramatically speed up training convergence and to reduce errors. Up to now though, they have to be designed a-priory and utilize structural knowledge of the underlying network. We propose a dynamic scheme where the participating workers self organize via p2p network with the goal of forming groups of similar optimization step times. To achieve this we utilize native features of distributed hash tables.

1 Introduction

In the domain of federated learning (FL) where data should not escape the premises of the device or the silo, efficient communication between the workers is paramount. In general the types of algorithms [4], [6] that operate on the local data also require sharing the results of each step with all or just their neighboring nodes. A characteristic example would be the family of gradient descent algorithms like stochastic gradient descent (SGD) or the distributed variant (dSGD). Thus the topology of the network, in the sense of which neighbors are available to each worker, can drastically affect the efficiency of the computation. Recent works attempting to exploit the topology of the network [9], [14] have been very successful in achieving even order of magnitude faster convergence of dSGD runs. They rely on knowledge of the underlying network to create efficient overlay topologies. This works well for the cross-silo FL case because the network is mostly static. But it also requires a-priori knowledge of it, and once the overlay is created, it is not easy to change.

We will attempt to resolve this limitation by leveraging tools from the field of peer to peer (p2p) research and more specifically, distributed hash tables (DHTs), thus creating a dynamic network structure that will still be throughput efficient. This approach would also not require any prior knowledge of the underlying network, while at the same time achieving progressively better connectivity. Finally our approach will have fault tolerance built-in since the system we will build on is designed to accommodate node failures.

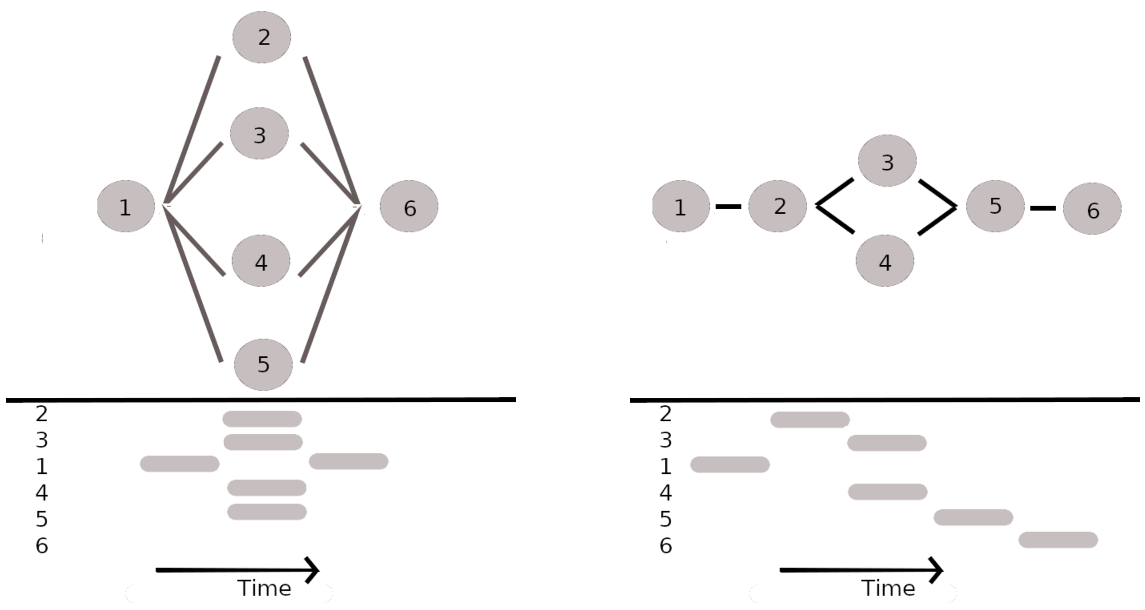

Figure 1: two different configurations that demonstrate how parallelism achieves better convergence time

One important observation that we make is that for every step of the algorithm the node requires some time to perform its local update T_l and some time to communicate with its neighbors T_n. This means that nodes are in lockstep, as is the case for many parallel discrete event based algorithms. Thus by using T_l as an input for a locality metric on the DHT keys, the formed neighborhoods impact how efficiently the algorithm converges. The key insight to our approach is that by *using the DHT as our neighbor discovery and routing platform, we will form groups of similar ΔT that will be connected in parallel as shown in Figure 1.* This should allow for optimal global convergence time of the algorithm. In addition, proximity neighbor selection (PNS) can be employed as well to minimize for the underlying network latency that exists within T_n.

The paper is structured as follows

- In *Related Work* we note important contributions in selecting topologies for FL tasks as well other uses of p2p concepts on them. We also list research on topologies and certain p2p networks that have promising properties if adopted for our needs.

- In *Design* we define the communication model and which properties our solution hopes to satisfy. By reasoning about the nature of the dSGD algorithm we notice how topology can drastically affect convergence times. We draw analogs from similar problems in related fields to propose modifications on DHTs that would make them efficient overlays for FL tasks. Finally we will investigate a setup where a part of the computation for each round is expected not to finish, either by stragglers or nodes that failed. We will design around such a setup and observe that it doesn't only dramatically increase convergence times for the FL task operating on most of the data, but also acts as a fault tolerance mechanism.

- In *Implementation* we present the design of our discrete event simulator that helps evaluate the hypotheses that have been made. Initial runs are performed that show the impact of local update time as a neighborhood metric.
- In *Evaluation* we will demonstrate the impact our dynamic topologies have on the convergence time of the simulated FL task completion times. We will also attempt to address some obvious shortcomings of the naive initial design proposed.

And the main contributions of the paper are

- a way to instruct similarly paced workers to form an overlay network using a DHT
- a way of exploiting fast local convergence to achieve faster global convergence by discarding only a small part of the total computation, in a controlled fashion
- a reliable inter-node communication scheme that by using the above position provides resilience for the whole network.

2 Related Work

As it has been observed in the seminal survey work [7] some of the core assumptions in Federated Learning, like the cross-silo,or cross-device setups could be challenged by utilizing p2p networks. It is also notes that topology is a crucial factor. The first approach to exploit the topology was MATCHA [14] was based on matching decomposition sampling, that reduced the communication delay per iteration of the dSGD algorithm. A throughput optimal topology design was also suggested [9] and where the input is the connectivity graph of the participating nodes. From there an overlay is designed which maximizes the network throughput of the system and thus reduces the time it take for runs of the dSGD algorithm to converge. The metric that is taken is called the mean cycle time and depends on the total delay between to consecutive steps of the algorithm called the `mean cycle time` (MCT). Their delay model incorporates both T_n, T_l of each step. Before that other p2p decentralized SGD approaches have been noted [3] but not really considering the topology as an optimizing factor.

Another important body of work that we investigated was the structure of topologies generated by p2p networks and how they fit in providing sufficient overlays for FL tasks. Hypercube-like routing schemes like Pastry [10] or ring-based like Chord [12] can, with proper modifications, offer efficient DHTs to build our dynamic topologies on top. Skip-graphs [2] also seem to satisfy many of our requirements. Finally we note that fat tree topologies [8] are investigated for their provably efficient communication properties and their successful deployment in many HPC setups. On the subject of fault tolerance we investigated how to connect traditional p2p fault tolerant routing as well as networking fault-tolerance schemes [11].

3 Design

3.1 Problem Setup

Distributed stochastic or mini-batch gradient descent algorithms like cooperative SGD [13] work in lockstep between performing local model updates, and communicating with their neighbors. In the case of cooperative SGD there can be κ steps of local updates between successive neighbor communications, here for generality we can consider that a neighbor update will happen at every iteration. Thus at an arbitrary iteration s of the algorithm at a node i must first receive possible updates from the neighborhood, then operate on its mini-batch and finally upload the results back to the neighborhood. If we ignore the upload time for the neigbors next step since it is also encoded as a download time from the previous step we can express the total delay as.

$$\Delta T^s_{(i,j)} = \max_j(T^{s-1}_j) + T^s_i$$

In the above setup all possible network induced delays are encoded in the neighborhood term. For now this keeps the model simple and we focus on the local delay of each node T^s_i. It is evident that different choices of neighborhoods can affect the whole network convergence time.

As seen above in Figure 1 if the *parallel group* shares approximately the same Δt it should finish its computation step at approximately the same time. But if part of it is a slower node, then its ΔT will be shared by the whole group. Thus an important aspect of our system will be first the discovery of similarly paced nodes around us and the formation of `parallel groups` with them. The model that we will operating considers the local computation time to be significantly more than internode messaging over the network. Another important point here is that the size of messages exchanged will be small. Only the results of the SGD computation (the calculated local gradients) are communicated, the actual data remain local to the node that processes them, an important feature of Federated Learning systems.

3.2 DHTs as topology managers

We are seeking a completely decentralized solution that can start from any initial topology and quickly converge to a throughput optimal one, without any a-priori knowledge of the network. By employing locality-preserving hashing with ΔT as the key, we will gain naturally from the DHT the ability to create neighborhoods of similarly paced nodes. Due to the way we are planning to use it a DHT that supports at least basically range queries is needed. For this purpose we considered three possible candidates.

- Pastry can be used since each node maintains neighbor lists. A node interested in getting a list of participants within a ΔT of its neighborhood look in its neighbor list to see if that range is completely covered by the existing participants, or otherwise push out a message with associated with TTL to both directions that would accumulate the nodes in that range. Once the TTL expires or the range is exceeded the last owner of the accumulation reports it back to the node.

- Chord with added predecessor/successor lists to each node, or some variant of the algorithm like BNN-Chord can be used [5].

- Skip-graphs [2] which were designed to address this specific problem can also be employed. Another property they have is that they can tolerate removal of large groups of nodes (up to a fraction of $O(\frac{1}{logN})$ without becoming highly disconnected. They could be desirable as the need for removal and reinsertion of nodes with their updated metrics as the topology improves, translates to high churn.

Considering the above options, we decided to move forward using Pastry as a DHT. The reason for that is that pastry, logically separates the Neighborhood set, the Routing table and the Leaf set. In traditional pastry, the Leaf set and the Routing table are keyed on the actual DHT key. But the Neighborhood set can also take other parameters into account (like RTT, or some other form of network proximity). This flexible design seems like a good choice for our DHT since by keying only on time to complete an SGD round, we can create neighborhoods that are non preferable in other metrics.

We believe that the frequent rekeying argument that would benefit the choice of skip-graphs, can be controlled by updating of the key only once i local SGD steps have been performed. Proven pace-steering techniques that are used in other FL setups [4] to address this exact problem can be used here to ensure that our DHT will never experience a large number of nodes attempting to remove and reinsert themselves.

3.3 Inter-Neighborhood Connectivity

We have seen that in the presence of diversely paced nodes we want to form neighborhoods based on that pace. The question arises of how should we interconnect these between them. In every such neighborhood there will be a minimum (the fastest) and a maximum valued node on ΔT (the slowest). For arguments we saw above, that whole group can only be as fast as its slowest node. Therefore the connection to the directly slower group above it should only happen from that node. Equivalently the connection to a faster group below, should happen from the fastest node. This is a well known interconnectivity scheme named Fat-Tree [8] that is used in high throughput packet switching.

It is possible to manipulate the locality function of the DHT to enforce a level like structure. By changing the hashing function from the identity on ΔT to something of the form

$$H(\Delta T) := \lfloor log(f(\Delta T, globalproperties)) \rfloor$$

Work still remains in constructing exact form of the function, taking into account the desired nodes per level, as well as levels in the whole range. That would enforce the desired tree like structure on the ring.

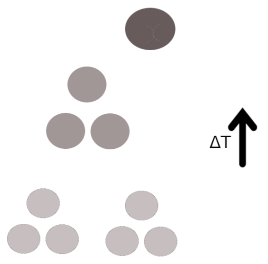

Figure 2: Fat tree topology on increasing ΔT.

3.4 Most stragglers left behind scheme

In the way we have presented the computation till now, it should be evident that we can not get significant gains on the final convergence time. Since some delaying nodes will still be connected, then their neighbors will have to wait for them, before starting the next round. During that time the rest of the network can not move to next iteration. This is why we propose to only require a percentage of the total nodes to complete the computation. The nodes that are ready will be signalling their upstream (up in the sense of delay) neighbors eagerly and once a number of nodes relative to the size of the group has completed their iteration cancellation signals will be sent to the stragglers and a new round will begin. This approach has the benefit of unifying stragglers with potentially failed nodes and utilizing a common strategy.

3.4.1 Failure Handling

An established way to increase resiliency is to provide more possible links between nodes as mentioned also in [11]. In that sense, more than one overlay networks can coexist, to account for possible link failures. In the case of a node failure, there is not much that our system can do, but since we don't require all the nodes to finish the computation, data points from other SGD participants will attempt to cover for that loss. For this reason, we chose Pastry as our initial DHT attempt. By being able to maintain at the same time neighbors in two different metrics, we can chose to forward a message from a potentially failed link from the other set. By the design of our network most a message generally will leave a node travelling towards a node that is still processing its local data. We want

- our message to reach more than one neighbors in case one of them crashes.
- our computation to not be double counted.

To achieve this we will create a similar technique to a server-shim as it is mentioned in [1] or dynamic routing as mentioned in [11]. For a message to leave a node, it has to get the receipt acknowledgement of more than one "upstream" nodes. Only then the message will be consider delivered for that node. As for the double counting, the message which is being propagated has to be able to evaluate to a prefix sum at each node. When a message is received in node (B) from node (A) it is opened up and decision for each prefix it taken. if a prefix coming from (A)s message has been seen already, *and* the (A) message containing the seen prefixes at node (B) successfully transmitted to more than one upstreams, then this prefix is discarded. If the prefix in (A)s message has not been seen and (B)s message hasn't been transmitted, we add it to our outgoing prefix sum for our upcoming send action. If we have sent a message yet we receive from (A) a prefix we have not in our records we propagate it (always using our replicating server-shim like scheme). At any given moment if a node manages to gather a satisfactory number of collected prefixes, a message to all the participants who we haven't contacted yet (a very small amount of the network) is sent out to signal them that they can stop their computation and move to the next round.

3.5 On incorporating network related delays

For simplicity we have kept our model intentionally vague. It seems there is only one form of delay that dictates the pacing of the nodes and we have presented it to be the ΔT of the local step. But in the original model we also have the network delays between the neighbors. The observation to be made here is that we really care for the largest of the two delays whichever that is. The other one can be incorporated as an appropriately scaled proximity network selection term. That means in our resulting routing key on the DHT can have its high order bits set by the driving delay on each node and use lower order bits for the remaining one.

4 Implementation

We have designed a completely event based simulator that generates various graphs, can apply delay distributions on the nodes and then perform the required amount of iterations on them. The code is already functional, yet a lot of work remains to incorporate the proper topology constructors. The simulator is written an golang and can work with event deltas up to the millisecond range, and with thousands of concurrently running nodes, easily in commodity hardware. To date we have

- a way to generate and interconnect overlay connectivity graphs, using linear, parallel,random and tree-like structures.
- a way to annotate them with workload characteristics (for example expected local gradient step time, and delta)
- the ability to perform a simulation run for arbitrary iterations on that network and collect statistic.

Our simulator can export data to JSON and the dot graph language, as well as import from these forms for ease of automation. Digraphs are supported natively and actually although not seen in this report, the engine treats graphs as directed.

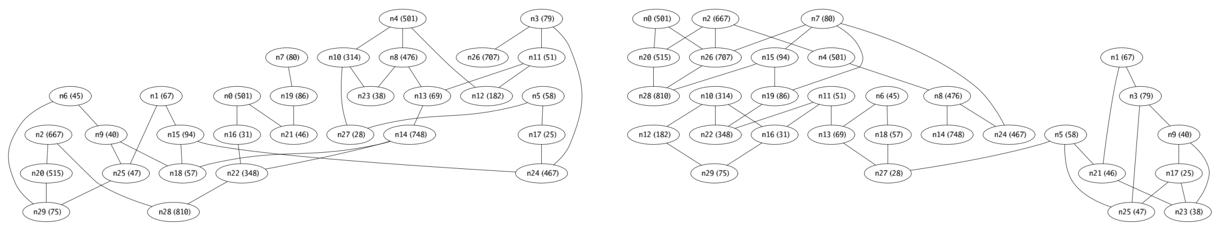

Figure 3: Two randomly generated overlays with average delays marked in parentheses. On the right a randomly wired one. On he left a fat-tree overlay.

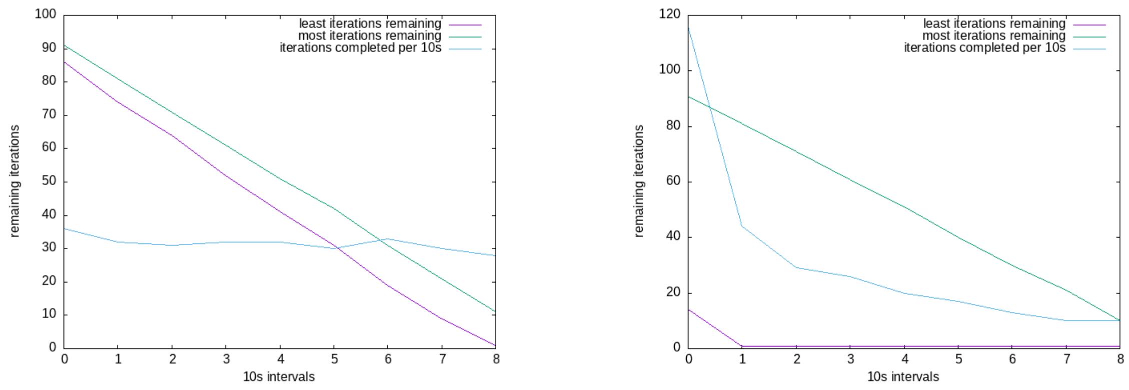

Figure 4: On the left the convergence profile of the random overlay versus the one of the fat tree on the right

5 Evaluation

Since this is highly work in progress we will just demonstrate some very preliminary results of the simulator and the effect of interconnectivity on the convergence. In the below example the overlay graphs are kept small for visualization purposes but we can easily run experiments on tens of thousands of nodes.

Now we can see how the convergence behaves between these two graphs when simulated for 100 iterations. In the simulation we do not employ the ”Most stragglers left behind” strategy. Therefore it is expected for the total time of `All` nodes of our simulation to be very close in both overlays. In the case of the fat tree structure for 20 experiments the average time of convergence was 1m 38,2 seconds seconds while for the random overlay the same average was 1m 40,5 seconds which for 100 iterations run gives us minor speedup of approximately 2%. The convergence behavior though is drastically different. In the following diagrams we can see in each at 10s snapshots, the remaining iterations for fastest converging node, for the slowest converging node, and also the average iterations that were completed for that 10s interval. From the above diagram we believe that if we accept a computation loss of a v small part of the network, the speedup we will gain will be logarithmic to the number of remaining nodes. As we also saw above , by expecting a small amount of the computation to fail too, we gain fault tolerant properties for the remaining participants. We intend to perform more runs and also investigate how the same properties scale according to the number of nodes and the number of iterations. Once a good overlay topology based on local delay is established we will begin the implementation of the DHT variant of it in the sense of implementing the key function that would shape it accordingly.

6 Closing Remarks and Future Work

As we see the choice of delay as parameter for the formation of neighborhoods on an overlay graph seems promising. It allows significantly larger throughputs in the initial runs of an algorithm and slight overall wall time improvement. If we are interested in having most of our nodes converge faster then it makes a great choice for an overlay graph. We also see how the delay, being a local property, combined with a DHT can allow a node to detect neighbours based either on that property, either on network proximity, or both, and we have sketched a way to move forward with an implementation. We still need to implement the DHT for which we have began work that would require approximately 2 months for completion. Apart from the DHT the straggler strategy needs to implemented in the simulator as well as the reliable message delivery mechanism.

References

[1] Remzi Can Aksoy and Manos Kapritsos. Aegean: Replication beyond the client-server model. In *Proceedings of the 27th ACM Symposium on Operating Systems Principles*, SOSP '19, page 385–398, New York, NY, USA, 2019. Association for Computing Machinery.

[2] James Aspnes and Gauri Shah. Skip graphs. *ACM Transactions on Algorithms*, 3(4):37, November 2007.

[3] Aurélien Bellet, Rachid Guerraoui, Mahsa Taziki, and Marc Tommasi. Personalized and private peer-to-peer machine learning. In Amos Storkey and Fernando Perez-Cruz, editors, *Proceedings of the Twenty-First International Conference on Artificial Intelligence and Statistics*, volume 84 of *Proceedings of Machine Learning Research*, pages 473–481. PMLR, 09–11 Apr 2018.

[4] Keith Bonawitz, Hubert Eichner, Wolfgang Grieskamp, Dzmitry Huba, Alex Ingerman, Vladimir Ivanov, Chloé Kiddon, Jakub Konecný, Stefano Mazzocchi, H. Brendan McMahan, Timon Van Overveldt, David Petrou, Daniel Ramage, and Jason Roselander. Towards federated learning at scale: System design. *CoRR*, abs/1902.01046, 2019.

[5] F. Chao, H. Zhang, X. Du, and C. Zhang. Improvement of structured p2p routing algorithm based on nn-chord. In *2011 7th International Conference on Wireless Communications, Networking and Mobile Computing*, pages 1–5, 2011.

[6] Chaoyang He, Tian Xie, Zhengyu Yang, Zijian Hu, and Shuai Xi. Decentralized federated multi-task learning and system design, 2019.

[7] Peter Kairouz, H. Brendan McMahan, Brendan Avent, Aurélien Bellet, Mehdi Bennis, Arjun Nitin Bhagoji, Keith Bonawitz, Zachary Charles, Graham Cormode, Rachel Cummings, Rafael G. L. D'Oliveira, Salim El Rouayheb, David Evans, Josh Gardner, Zachary Garrett, Adrià Gascón, Badih Ghazi, Phillip B. Gibbons, Marco Gruteser, Zaïd Harchaoui, Chaoyang He, Lie He, Zhouyuan Huo, Ben Hutchinson, Justin Hsu, Martin Jaggi, Tara Javidi, Gauri Joshi, Mikhail Khodak, Jakub Konecný, Aleksandra Korolova, Farinaz Koushanfar, Sanmi Koyejo, Tancrède Lepoint, Yang Liu, Prateek Mittal, Mehryar Mohri, Richard Nock, Ayfer Özgür, Rasmus Pagh, Mariana Raykova, Hang Qi, Daniel Ramage, Ramesh Raskar, Dawn Song, Weikang Song, Sebastian U. Stich, Ziteng Sun, Ananda Theertha Suresh, Florian Tramèr, Praneeth Vepakomma, Jianyu Wang, Li Xiong, Zheng Xu, Qiang Yang, Felix X. Yu, Han Yu, and Sen Zhao. Advances and open problems in federated learning. *CoRR*, abs/1912.04977, 2019.

[8] Charles E. Leiserson. Fat-trees: Universal networks for hardware-efficient supercomputing. *IEEE Trans. Comput.*, 34(10):892–901, October 1985.

[9] Othmane Marfoq, Chuan Xu, Giovanni Neglia, and Richard Vidal. Throughput-optimal topology design for cross-silo federated learning, 2020.

[10] A. Rowstron and P. Druschel. astry: Scalable, distributed object location and routing for large-scale peer-to-peer systems. In *Middleware 2001*, 2001.

[11] F. Sem-Jacobsen, O. Lysne, J. Duato, and T. Skeie. Dynamic fault tolerance in fat trees. *IEEE Transactions on Computers*, 60(04):508–525, apr 2011.

[12] Ion Stoica, Robert Morris, David Liben-Nowell, David R. Karger, M. Frans Kaashoek, Frank Dabek, and Hari Balakrishnan. Chord: A scalable peer-to-peer lookup protocol for internet applications. *IEEE/ACM Trans. Netw.*, 11(1):17–32, February 2003.

[13] Jianyu Wang and Gauri Joshi. Cooperative SGD: A unified framework for the design and analysis of communication-efficient SGD algorithms. *CoRR*, abs/1808.07576, 2018.

[14] Jianyu Wang, Anit Kumar Sahu, Zhouyi Yang, Gauri Joshi, and Soummya Kar. MATCHA: speeding up decentralized SGD via matching decomposition sampling. *CoRR*, abs/1905.09435, 2019.