Decentralized Learning Made Easy with DecentralizePy

Akash Dhasade
EPFL

Anne-Marie Kermarrec
EPFL

Rafael Pires
EPFL

Rishi Sharma*
EPFL

Milos Vujasinovic
EPFL

Abstract

Decentralized learning (DL) has gained prominence for its potential benefits in terms of scalability, privacy, and fault tolerance. It consists of many nodes that coordinate without a central server and exchange millions of parameters in the inherently iterative process of machine learning (ML) training. In addition, these nodes are connected in complex and potentially dynamic topologies. Assessing the intricate dynamics of such networks is clearly not an easy task. Often in literature, researchers resort to simulated environments that do not scale and fail to capture practical and crucial behaviors, including the ones associated to parallelism, data transfer, network delays, and wall-clock time. In this paper, we propose DECENTRALIZEPY, a distributed framework for decentralized ML, which allows for the emulation of large-scale learning networks in arbitrary topologies. We demonstrate the capabilities of DECENTRALIZEPY by deploying techniques such as sparsification and secure aggregation on top of several topologies, including dynamic networks with more than one thousand nodes.

CCS Concepts: • **Networks** → **Programming interfaces**; • **Computing methodologies** → *Distributed algorithms*; *Machine learning algorithms*; • **Computer systems organization** → **Peer-to-peer architectures**.

Keywords: decentralized learning, middleware, machine learning, distributed systems, peer-to-peer, network topology

ACM Reference Format:
Akash Dhasade, Anne-Marie Kermarrec, Rafael Pires, Rishi Sharma, and Milos Vujasinovic. 2023. Decentralized Learning Made Easy with DecentralizePy. In *3rd Workshop on Machine Learning and Systems (EuroMLSys '23), May 8, 2023, Rome, Italy.* ACM, New York, NY, USA, 8 pages. https://doi.org/10.1145/3578356.3592587

*Corresponding author: first.last@epfl.ch

EuroMLSys '23, May 8, 2023, Rome, Italy

ACM ISBN 979-8-4007-0084-2/23/05…$15.00
https://doi.org/10.1145/3578356.3592587

1 Introduction

There has been a shift in machine learning (ML) training, which is now taking place at the location where data is generated, instead of the earlier method of first moving the data to be processed in data centers. Federated learning (FL) [26] and decentralized learning (DL) [23] have become prominent collaborative training methods that prioritize privacy, by solely exchanging updates of the model being trained. In FL, training is orchestrated by a central server where the server broadcasts the global model to participating nodes and later aggregates the updates received back from them. Conversely, in DL, nodes are connected in a fully decentralized communication topology. Rather than relying on a server, nodes in DL train on their local datasets and share the resulting models with neighboring nodes. The aggregated models obtained from these exchanges lead to a converged global model at the end of the DL training process. While FL has already received more attention from academia and industry [9, 11, 17, 37], DL is also recently gaining a lot of traction [18–20, 23].

The popularity of FL has been partially enabled by the availability of several frameworks for simulation or emulation [1, 7, 14, 22, 25, 30, 39], which allow researchers to quickly implement and assess novel FL algorithms. Support for DL configurations, on the other hand, is rare and limited in these existing frameworks [14]. FL frameworks typically offer abstractions for client-server interactions which are not suitable for DL, since nodes in DL communicate only with their immediate neighbors. These limitations call for frameworks that provide abstractions involving peer-to-peer communication channels and topology manipulation for collaborative ML training.

DL research has often focused on exploring the impact of topology on learning performance [5, 33], which has led to simulations using various static and dynamic topologies during training [19]. Another important area of DL research targets to improve communication efficiency through compression, which reduces the number of model parameters exchanged between DL nodes [3, 24, 32]. This includes techniques such as sparsification [3] and quantization [2], which are either not readily available in FL frameworks, or incompatible with DL. The same happens in secure aggregation [10], which imposes additional challenges when deployed in DL training. DECENTRALIZEPY offers these capabilities, empowering researchers to explore and innovate on

DL research without being constrained by the limitations of existing FL frameworks.

Although we are not exhaustive in implementing all aspects of DL systems, we provide, along with reference implementations, a modular and extensible framework that aims at easing the prototyping and deployment of such systems. We demonstrate this flexibility with various topologies, sparsification, and secure aggregation techniques in DL training. Our results provide insights to encourage more research on these areas in order to improve the practicality of DL.

Contributions

- We present DECENTRALIZEPY, a novel decentralized learning framework, which allows researchers and practitioners to experiment with the components of DL systems and deploy them in real-world settings.
- We showcase how DECENTRALIZEPY can be leveraged to assess the effects of several topologies, dynamic networks, sparsification techniques, and secure aggregation in DL systems.
- DECENTRALIZEPY is modular, easily extendable, and open-source [31] under *MIT License*.

The remainder of this paper is organized as follows: Section 2 describes the framework internals and Section 3 showcases its usage in several scenarios. We survey related work in Section 4 and conclude in Section 5.

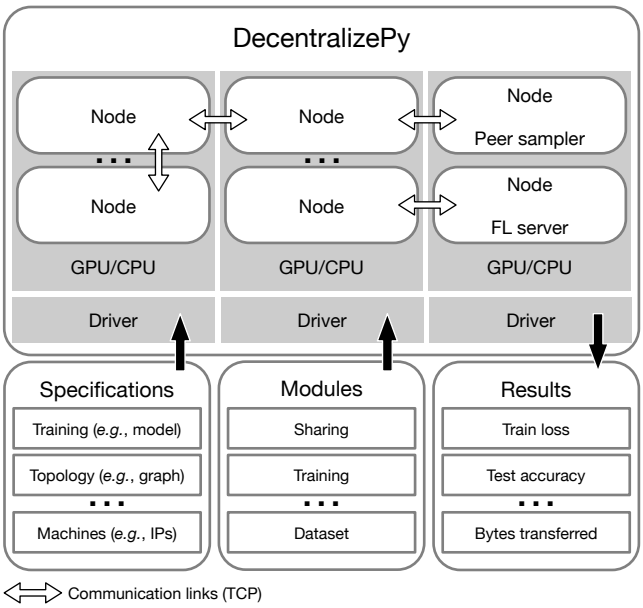

Figure 1. Overview of the DECENTRALIZEPY framework. Each node along with its driver may run on a separate machine. The driver takes as input the specifications and modules. The node dynamically loads the specified modules. Results are dumped locally and later aggregated. Nodes can be specialized to perform different tasks. In conventional DL, we would have only basic Node(s). To emulate FL, a node can be modified to coordinate the training, shown as the FL server.

2 DECENTRALIZEPY

2.1 Design overview

Works in DL research are mostly evaluated in simulated environments. These simulations are either done in scenarios where a single machine simulates all nodes (*i.e.*, not scalable), or in cluster settings over MPI (*i.e.*, constrained to the local area network (LAN)) [5, 33, 34, 38]. The design of DECENTRALIZEPY takes into account two key needs: the ability to quickly develop research prototypes, as well as actually deploying large-scale DL systems. We elaborate next on these design goals.

Modularity. DL systems consist of multiple components, and conducting research in this field requires adjusting and testing them. For generality, it is often necessary to evaluate different datasets and models when building learning systems. In decentralized training, the overlay topology, *i.e.*, the way nodes are connected, is especially critical for achieving satisfactory model convergence [4]. Finally, the protocol itself, *i.e.*, who to communicate with, what is the message content, and how to aggregate the received parameters, varies among systems. To facilitate the implementation of new DL systems and experimentation with various ML aspects, including datasets, models, and topologies, DECENTRALIZEPY incorporates loosely coupled modules with an object-oriented design. Section 2.2 provides a more detailed explanation of these modules.

One-node one-process. In order to achieve rapid prototyping, researchers commonly favor simple implementations built from scratch. In DL systems, this often means simulations on a single machine or in a cluster environment [5, 33, 34, 38]. Real-world deployment of such systems would however imply numerous nodes running on geographically-distant machines. Unfortunately, DL systems specifically designed for cluster environments are difficult to configure and scale beyond the LAN. DECENTRALIZEPY has a *one-node as one-process* design principle, so that we are able to scale and measure realistic system metrics at the granularity of a node (or process). In other words, one DL node is represented as a single process. These nodes communicate over network sockets and do not distinguish processes on the same or different machines. This means that the same testbed can run in a cluster environment or on real-world machines over wide area networks (WANs) by just configuring the IP address information. This approach simplifies the configuration of system parameters like the number of processes per node (according to, for instance, the number of CPU cores), network bandwidth, latency, and packet drop, enabling practitioners to study their systems in detail. In addition, measurements of system metrics such as data transferred, memory usage, and CPU time become easier to collect on each node. Furthermore, the emulation and deployment can reveal behaviors that would not show

up in simulations. These benefits streamline the assessment of the performance and scalability of DL systems.

2.2 Architecture

In this section, we describe DECENTRALIZEPY modules and how they can be customized to develop new DL systems. Figure 1 shows the architecture of the framework and how different modules are integrated.

Node. The `node` module acts as the skeleton code that performs the DL task by instantiating and calling the methods of the rest of the modules. An object of a sub-class of the `node` module is instantiated when a DL process starts. A `node` can be designed to perform a variety of tasks from being a DL client to a FL server or a centralized peer sampler. The `node` module provides complete flexibility in creating and decoding messages to be exchanged with peers. To get practitioners started with DECENTRALIZEPY, the framework is already equipped with the implementations of DL clients, a centralized peer sampler, a parameter server, a FL server, and secure aggregation clients as nodes.

Graph. The `graph` module manages the overlay network, which constrains the communication of nodes to only immediate neighbors. This overlay graph can be modified at run time by the node, hence supporting both static and dynamic topologies. The topology can be read from a graph file having edges or an adjacency list. With this, we support swift switching of topologies, which could be generated by external libraries, during experimentation. A graph file specifying the topology is shown as the *topology specification* in Figure 1.

Model. This is a lightweight module inheriting the model class of the underlying ML framework. The main purpose of having a `model` module is to store additional states. For instance, this module allows the developer to store past gradients or how much the learning parameters changed in the last iteration. This is especially convenient for some sparsification algorithms. Since the models are quite specific to the datasets, these are implemented as part of the `dataset` modules.

Dataset. ML frameworks must support a diverse range of datasets and models. The `dataset` module of our framework provides this for a variety of learning tasks over multiple datasets and models, seamlessly integrating with the other modules. Among the tasks it performs, we highlight: *(i)* reading the train and test sets; *(ii)* partitioning the datasets among nodes; *(iii)* evaluating the performance of a trained model on the test set; and *(iv)* specific model implementations. DECENTRALIZEPY includes six datasets from the LEAF FL benchmark [11] and CIFAR-10 [21] with both independent and identically distributed (IID) and non independent and identically distributed (non-IID) data partitioning.

Training. The `training` module performs local training steps of the model on the given training set. The optimizer and loss function for the learning tasks are provided as part of the training specifications shown in Figure 1. Having access to the instances of both `model` and `dataset` modules, the `training` module can modify the additional state variables in these modules for various purposes, *e.g.*, prioritizing certain weights during model compression.

Sharing and Communication In DL, the nodes interact by exchanging messages. The `sharing` module decides the contents of these messages and the aggregation procedure. For model sharing, the messages would contain serialized parameters and the aggregation scheme will average the received models. In the presence of sparsification (model compression), the messages would contain a subset of model parameters, and the aggregation scheme needs to account for missing parameters. Implementations of sparsification schemes such as random sampling, `TopK` [3], and CHOCO-SGD [20] are available as `sharing` modules in DECENTRALIZEPY. For data-sharing architectures [12], in turn, the `sharing` module would include raw data in the messages, and the aggregation procedure would append the received data to the local dataset. These messages are passed on to `communication` by the node to send them to the correct recipients. DECENTRALIZEPY also has an existing implementation of `communication` using ZeroMQ [8] over TCP.

Mapping, Compression, and Utils There are some auxiliary modules to support the framework: (1) `Mapping`, (2) `compression`, and (3) `utils`. To support both cluster environments and real-world deployment, `mapping` associates

```
1  from decentralizepy.node.Node import Node
2
3  class DLNode(Node):
4      def run(self, iterations, training, dataset,
           sharing, graph, communication):
5          for round in range(iterations):
6              training.train(dataset)
7              msg = sharing.get_message()
8              neighbors = graph.get_neighbors()
9              communication.send(neighbors, msg)
10             rcv = communication.receive_from_all()
11             sharing.average(rcv)
12             dataset.test()
```

Figure 2. A Python code snippet to demonstrate a simple DL node using the modules of DECENTRALIZEPY colored in red. The node repeatedly trains its model on the local dataset (line 6), exchanges the model with the neighbors (lines 7-10), aggregates the models (line 11), and evaluates the average model on the test set (line 12).

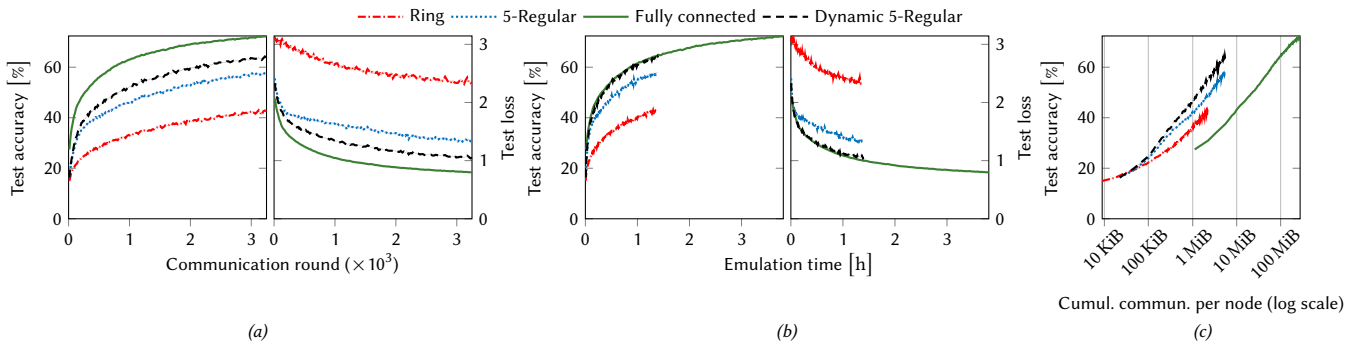

Figure 3. Performance of 256-node DL across three topologies and a dynamic 5-regular graph. *(a)* The denser the topology, the better the accuracy: fully connected > d-regular > ring. They all run for the same number of communication rounds. *(b)* When considering emulation time, fully connected takes the longest to perform the same number of rounds. *(c)* Denser topologies incur significantly more communication costs. We observe that d-regular graphs offer a favorable tradeoff between accuracy, communication, and emulation time compared to *ring* or *fully connected*. Dynamic d-regular surprisingly matches the convergence of fully connected across time (b) at significantly lower communication cost (c).

nodes with particular machines. `Compression` module packages general-purpose compression algorithms for floating-point and integer lists. `Utils` contains commonly used functions across the framework like Python dictionary manipulations and command-line argument parsing.

DECENTRALIZEPY allows the developer to instantiate a node and provide the implementations of the other modules separately, as shown in Figure 1. Each module provides a base class that defines the interface for the module. New implementations of modules can extend the base class and legacy implementations may be ported by wrapping them in the module interface. These implementations are dynamically loaded to streamline the process of substituting the implementations for rapid prototyping. Furthermore, since DECENTRALIZEPY is fully-decentralized, the nodes can have heterogenous datasets, different sharing strategies, optimizers, and learning rates for full flexibility. Figure 2 shows how to write a simplified decentralized learning node using DECENTRALIZEPY. The functions of the other modules (*e.g.*, `sharing`) invoked here can be overloaded to customize the system. Given the decentralized nature of the nodes, each of them locally writes logs and results in JSON files. To compute aggregate statistics of the system, we collect and process the results in a single machine at the end.

2.3 Implementation

Written in Python v3.8, the modules span a total of over 10 000 lines of code. Additionally, the framework inherits core ML functionalities from PyTorch v1.10.2 [29]. DECENTRALIZEPY contains several implementations of modules to facilitate building DL systems as described earlier. It is important to note that these are just reference implementations for a quick start in developing new DL systems with the framework. We call for the support of the scientific community to improve DECENTRALIZEPY by adding more implementations of modules derived from recent and upcoming research.

3 Evaluation

In this section, we demonstrate the power of DECENTRALIZEPY by implementing changing topologies, state-of-the-art sparsification algorithms, secure aggregation in DL, and a scalability study.

3.1 Experimental Setup

The experiments are run on 16 hyperthreading-enabled machines equipped with 2 Intel(R) Xeon(R) CPU E5-2630 v3 @ 2.40GHz having 8 cores. Based on the experiment, we run 48, 256, or 1024 DL nodes (processes). Each process is constrained to one logical CPU core in the experiments. The nodes are oblivious about being on the same or different machines and communicate via TCP sockets using ZeroMQ [8]. We tuned the learning-rate for the basic stochastic gradient descent (SGD) optimizer without momentum, and use it for training along with cross-entropy loss. The DL clients used decentralized parallel stochastic gradient descent (D-PSGD) [23] over Metropolis-Hastings weights [36] for aggregation. We use CIFAR-10 [21] dataset with a 2-sharding non-IID data partitioning [26] which limits the number of classes per node to 4. In addition to CIFAR-10, we also use CelebA [11] for secure-aggregation experiments. The evaluation is done on the test-set of the respective datasets. We run each experiment 5 times with different random seeds and present the average metrics with a 95% confidence interval.

3.2 Topologies and Dynamicity

The performance of DL is heavily influenced by the communication topology [18]. We now demonstrate how DECENTRALIZEPY can flexibly simulate different topologies. We start by running DECENTRALIZEPY with 256 learning nodes on a ring, d-regular with degree 5, and fully-connected static topologies. The framework allows us to effortlessly change topologies for the experiments by only replacing the graph

file described in Section 2.2. To simulate dynamic topologies, DECENTRALIZEPY uses a centralized peer sampler which instantiates new topologies every round using the `graph` module. Any dynamic graph can be realized within the peer sampler which then notifies each node of its neighbors. We demonstrate a sample case where the peer sampler creates a random 5-regular topology every round.

Figure 3 shows the convergence plots with respect to the communication rounds, wall-clock time, and the cumulative bytes sent per node when run for the same number of communication rounds. As one would expect, fully connected has the highest accuracy and the ring has the lowest accuracy at the end (Figure 3 (a)). DECENTRALIZEPY also reveals that experiments with fully-connected topologies take nearly 3× more time when compared to the rest for the same number of communication rounds (Figure 3 (b)). This detail is often not evaluated in literature due to limited framework support. We observe that d-regular topologies represent a favorable tradeoff between the extreme topologies (Figure 3 (b) and (c)). Dynamic topologies perform much better than their static counterparts. Moreover, dynamic 5-regular topology achieves almost identical accuracies to fully-connected given the same time deadline while having 51× less communication cost. Therefore, research should focus more on dynamic topologies to study their tradeoffs in more detail.

3.3 Sparsification

DL systems use sparsification algorithms to reduce the number of bytes exchanged in the network [3, 24]. In sparsification, a communication budget specifies the percentage of model parameters shared by nodes with respect to full sharing, *i.e.*, sharing all parameters. We now showcase how DECENTRALIZEPY supports commonly-used sparsification algorithms as part of the `sharing` module. In full sharing, the basic `sharing` module generates a serialized parameter vector to be sent to neighboring nodes. For sparsification, in contrast, we modify basic `sharing` to generate serialized tuples of the indices and values of the parameters chosen to be shared. For the experiments, we consider a setup with a 5-regular topology of 256 learning nodes with a communication budget of 10%.

Figure 4 shows the test accuracy vs. communication cost of two sparsification algorithms: (1) random sampling, and (2) hyperparameter-tuned state of the art CHOCO-SGD [20] against the baseline of full sharing DL. The random sampling algorithm picks 10% random parameters every round for sharing. CHOCO-SGD uses sophisticated parameter-ranking and error-correction schemes to limit the loss of information due to sparsification. We observe that sparsification loses too much information and performs significantly worse than full sharing. This can be attributed to the complexity of the learning task. The convergence of sparsification algorithms drastically slows down in non-IID settings with a large number of nodes. To reach the same accuracy as both

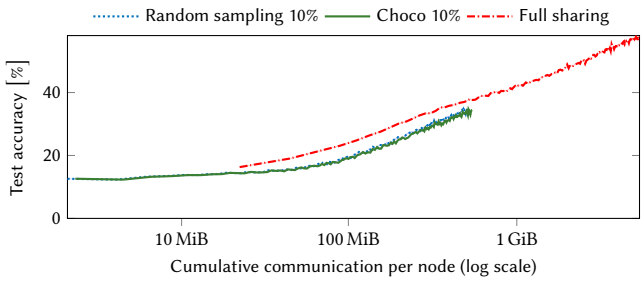

Figure 4. Performance of a 256-node DL comparing the sparsification algorithms of *random sampling* and CHOCO-SGD to *full sharing*. The communication budget is set to 10% and we run all algorithms for the same number of communication rounds. We observe that data non-IIDness and the scale of nodes significantly hurt the performance of sparsification algorithms. Under the same communication budget, full sharing tends to be robust and achieves higher accuracy.

sparsification algorithms, full sharing uses significantly less communication. Most works studying sparsification algorithms are often evaluated in IID settings with a limited number of nodes, resulting in overly optimistic estimations of the performance [15]. The use of DECENTRALIZEPY will enable researchers to study sparsification algorithms that are robust to difficult data distributions at scale.

3.4 Secure Aggregation

The well-established technique of secure aggregation [10], commonly used in centralized settings, ensures that participating nodes only have access to the aggregated model parameters, while keeping private the individual models of other training nodes. This is achieved through the masking of parameters where pairs of nodes add cancellable masks to their respective models before sharing. The receiver node upon aggregation gets the same aggregated model as one without secure aggregation. However, the masks prevent access to individual models. Through a non-trivial implementation [35], we show that DECENTRALIZEPY can also be leveraged to perform such secure aggregation in the DL setting. We design the core procedure as part of the `node` module of DECENTRALIZEPY and conduct experiments with 48 nodes on CIFAR-10 and CelebA datasets for 10 000 communication rounds.

Figure 5 shows the test accuracy vs. cumulative communication cost of DL with and without secure aggregation. Secure aggregation incurs approximately 3% more communication due to metadata (shared seeds for pseudo-random number generation and masks) in addition to the parameters. Furthermore, because masks and parameters are floating point numbers, there is a loss of precision which leads to 3% loss in accuracy.

3.5 Scalability

Finally, we perform a scalability study of DL by increasing the number of nodes from 256 to 1024. In this study, we

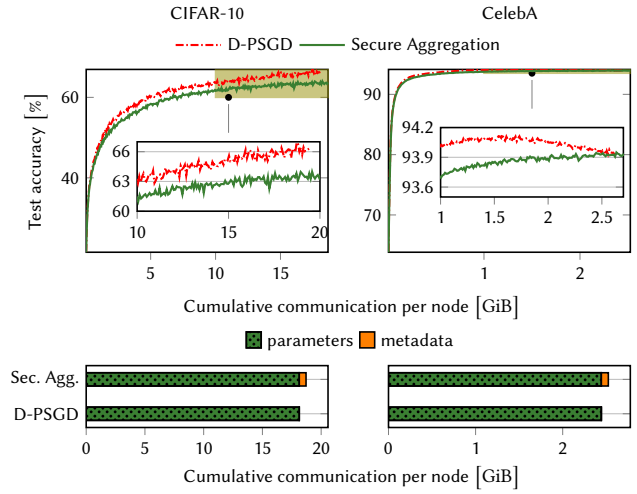

Figure 5. Performance of a 48-node DL comparing Secure Aggregation with standard DL without secure aggregation (D-PSGD). We observe that Secure Aggregation achieves comparable accuracy to D-PSGD on the CelebA dataset while it loses 3% absolute accuracy on CIFAR-10 (first row). The privacy guarantees of Secure Aggregation come at a modest cost in additional communication (second row).

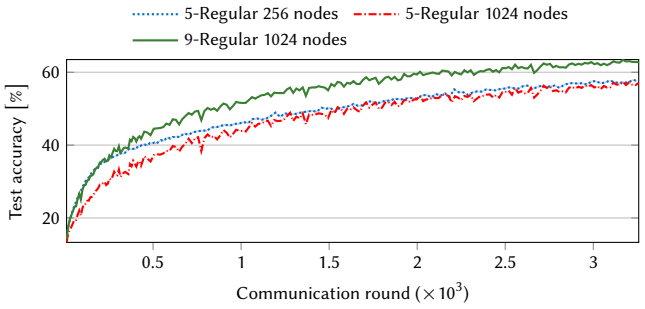

Figure 6. Performance of a 256-node and 1024-node DL setups with varying degrees of d-regular topologies. Since the total dataset size remains the same when the nodes are scaled, each node receives 4× fewer data samples in the 1024-node setup. We observe that 5-regular topology achieves the same performance in both 256- and 1024-node setups, suggesting that degree might be more influential than the number of data samples per node in DL. Further increasing the degree to 9 benefits accuracy by almost 6%.

particularly analyze the effect of the number of data samples per node and the degree of the node on the performance of DL. Since the underlying dataset size remains the same when the nodes are scaled, each node receives 4× fewer data samples in the setup with 1024 nodes compared to the setup with 256 nodes.

We report the evolution of test accuracy for 256-node 5-regular, 1024-node 5-regular, and 1024-node 9-regular topologies in Figure 6. It is worth noting that the final performance of the 5-regular topology with 1024 and 256 nodes remain almost the same, even with non-IID data and 4× fewer data samples in the 1024-node setting. Increasing the degree from 5 to 9 boosts the performance by 5.8 accuracy points on average. This suggests that the number of neighbors has a higher impact on learning than the number of data samples per node. Such novel insights are possible through advanced scalability studies, thanks to DECENTRALIZEPY. We highlight that DECENTRALIZEPY can run a much higher number of nodes relative to the relevant literature in decentralized learning systems [5, 20, 33, 34, 38].

4 Related work

Flower [7] is a framework for FL emulation. Like DECENTRALIZEPY, Flower scales to one thousand nodes training in parallel. It is however limited to the FL setup, *i.e.*, it relies on a central server that orchestrates the ML training by collecting and aggregating client models on every iteration. In addition, it does not offer seamless support for arbitrary communication topologies.

FedScale [22] is both a collection of realistic datasets (with natural non-IID and unbalanced data) and a runtime platform that supports cluster and mobile back-ends. In simulation mode, FedScale orchestrates execution requests over available resources (either single GPU/CPU or distributed) while keeping a *client virtual clock*. This way, it achieves shorter evaluation times in comparison to the simulated scenario and reports up to 10 000 training nodes in parallel on top of a cluster of 10 GPU nodes. In addition, FedScale incorporates system speeds and availability traces of mobile devices to achieve realistic heterogeneity simulations. Unlike DECENTRALIZEPY, it has no support for DL.

IPLS [28] proposes ML training that uses a decentralized filesystem as the communication channel. To reduce network traffic while ensuring some fault tolerance, they also propose model partitioning and replication of partitions across nodes. Each node only shares a particular set of layers (rather than the whole model) and each layer is potentially shared by several nodes. Its feasibility is yet to be confirmed, as authors show a very small deployment (50 nodes) with a simple learning task (MNIST) on top of IID data. Moreover, IPLS depends on the underlying IPFS [6] implementation and is therefore not flexible in terms of topology manipulation.

Kollaps [13] is a decentralized network emulator that allows for the manipulation of end-to-end communication properties between nodes. By shaping the network latency, bandwidth, packet loss, and jitter, they achieve very similar results of bare-metal deployments in geo-distributed setups. Kollaps is orthogonal and complementary to DECENTRALIZEPY, as it operates at the level of containers (*e.g.*, Docker) and orchestrators (*e.g.*, Kubernetes).

Closer to our proposal, FedML [14] is a Federated Learning framework that provides several baseline implementations of models and datasets for reproducible research and benchmarking. FedML supports distinct topologies, including decentralized ones. Since it uses MPI [27] for communication,

it is primarily targeted to cluster environments. DECENTRALIZEPY, in contrast, can be smoothly deployed in WAN and is more flexible than FedML in terms of dynamic topologies.

5 Conclusion

Research in DL systems requires efficient scaling to many nodes, ease of deployment, and a quick development cycle. In this paper, we present DECENTRALIZEPY, a DL framework designed to support these requirements. Its modular design facilitates the replacement of components with customized ones, allowing rapid prototyping and deployment.

We demonstrate the power, flexibility, and scalability of DECENTRALIZEPY through a series of experiments across different topologies, sparsification algorithms, and privacy-preserving secure aggregation. Using this framework, we were able to derive new insights concerning the effectiveness of dynamic topologies and the performance degradation of state-of-the-art sparsification algorithms on non-IID learning tasks at scale.

DECENTRALIZEPY has been used by over a dozen students for projects in DL. Their feedback has been valuable in improving the usability and efficiency of the framework. We expect the scientific community to also try and contribute to it. As future work, we plan to extend DECENTRALIZEPY with real-world traces of network traffic over Kollaps [13], client availability from FedScale [22], and decentralized peer sampling [16]. We also plan to integrate additional state-of-the-art DL algorithms, including quantization and topology construction. Finally, using the framework as a platform, the community can build communication-efficient and privacy-preserving DL systems that are robust to non-IID data distributions at scale.

References

[1] Martín Abadi, Paul Barham, Jianmin Chen, Zhifeng Chen, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Geoffrey Irving, Michael Isard, Manjunath Kudlur, Josh Levenberg, Rajat Monga, Sherry Moore, Derek G. Murray, Benoit Steiner, Paul Tucker, Vijay Vasudevan, Pete Warden, Martin Wicke, Yuan Yu, and Xiaoqiang Zheng. 2016. TensorFlow: a system for Large-Scale machine learning *(OSDI'16)*. https://www.usenix.org/conference/osdi16/technical-sessions/presentation/abadi

[2] Dan Alistarh, Demjan Grubic, Jerry Z. Li, Ryota Tomioka, and Milan Vojnovic. 2017. QSGD: Communication-Efficient SGD via Gradient Quantization and Encoding *(NIPS'17)*.

[3] Dan Alistarh, Torsten Hoefler, Mikael Johansson, Sarit Khirirat, Nikola Konstantinov, and Cédric Renggli. 2018. The Convergence of Sparsified Gradient Methods *(NIPS'18)*. https://proceedings.neurips.cc/paper_files/paper/2018/file/314450613369e0ee72d0da7f6fee773c-Paper.pdf

[4] Batiste Le Bars, Aurélien Bellet, Marc Tommasi, Erick Lavoie, and Anne-Marie Kermarrec. 2023. Refined Convergence and Topology Learning for Decentralized Optimization with Heterogeneous Data *(AISTATS'23)*. arXiv:2204.04452

[5] Aurélien Bellet, Anne-Marie Kermarrec, and Erick Lavoie. 2022. D-Cliques: Compensating for Data Heterogeneity with Topology in Decentralized Federated Learning *(SRDS'22)*.

[6] Juan Benet. 2014. IPFS - Content Addressed, Versioned, P2P File System. *CoRR* (2014). arXiv:1407.3561

[7] Daniel J Beutel, Taner Topal, Akhil Mathur, Xinchi Qiu, Titouan Parcollet, and Nicholas D Lane. 2020. Flower: A Friendly Federated Learning Research Framework. (2020). arXiv:2007.14390

[8] Luca Boccassi et al. 2023. ZeroMQ: An open-source universal messaging library. https://zeromq.org

[9] Keith Bonawitz, Hubert Eichner, Wolfgang Grieskamp, et al. 2019. Towards Federated Learning at Scale: System Design *(MLSys'19)*. https://proceedings.mlsys.org/paper_files/paper/2019/file/bd686fd640be98efaae0091fa301e613-Paper.pdf

[10] Keith Bonawitz, Vladimir Ivanov, Ben Kreuter, Antonio Marcedone, H. Brendan McMahan, Sarvar Patel, Daniel Ramage, Aaron Segal, and Karn Seth. 2017. Practical Secure Aggregation for Privacy-Preserving Machine Learning *(CCS '17)*. https://doi.org/10.1145/3133956.3133982

[11] Sebastian Caldas, Peter Wu, Tian Li, Jakub Konečný, H. Brendan McMahan, Virginia Smith, and Ameet Talwalkar. 2019. Leaf: A benchmark for federated settings. In *2nd Intl. Workshop on Federated Learning for Data Privacy and Confidentiality (FL-NeurIPS'19)*. arXiv:1812.01097

[12] Akash Dhasade, Nevena Dresevic, Anne-Marie Kermarrec, and Rafael Pires. 2022. TEE-based decentralized recommender systems: The raw data sharing redemption *(IPDPS'22)*. https://doi.org/10.1109/IPDPS53621.2022.00050

[13] Paulo Gouveia, João Neves, Carlos Segarra, Luca Liechti, Shady Issa, Valerio Schiavoni, and Miguel Matos. 2020. Kollaps: Decentralized and Dynamic Topology Emulation *(EuroSys '20)*. Article 23. https://doi.org/10.1145/3342195.3387540

[14] Chaoyang He, Songze Li, Jinhyun So, Mi Zhang, Hongyi Wang, Xiaoyang Wang, Praneeth Vepakomma, Abhishek Singh, Hang Qiu, Li Shen, Peilin Zhao, Yan Kang, Yang Liu, Ramesh Raskar, Qiang Yang, Murali Annavaram, and Salman Avestimehr. 2020. FedML: A research library and benchmark for federated machine learning. (2020). arXiv:2007.13518

[15] Kevin Hsieh, Amar Phanishayee, Onur Mutlu, and Phillip B. Gibbons. 2020. The Non-IID Data Quagmire of Decentralized Machine Learning. In *Proceedings of the 37th International Conference on Machine Learning (ICML'20)*. Article 408. http://proceedings.mlr.press/v119/hsieh20a/hsieh20a.pdf

[16] Márk Jelasity, Spyros Voulgaris, Rachid Guerraoui, Anne-Marie Kermarrec, and Maarten Van Steen. 2007. Gossip-based peer sampling. *ACM Transactions on Computer Systems (TOCS)* 25, 3 (2007), 8–es.

[17] Peter Kairouz, H. Brendan McMahan, Brendan Avent, Aurélien Bellet, Mehdi Bennis, Arjun Nitin Bhagoji, Kallista Bonawitz, Zachary Charles, Graham Cormode, Rachel Cummings, Rafael G. L. D'Oliveira, Hubert Eichner, Salim El Rouayheb, David Evans, Josh Gardner, Zachary Garrett, Adrià Gascón, Badih Ghazi, Phillip B. Gibbons, Marco Gruteser, Zaid 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, Hang Qi, Daniel Ramage, Ramesh Raskar, Mariana Raykova, 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. 2020. Advances and open problems in federated learning. *Foundations and Trends in Machine Learning* 14, 1–2 (2020). https://doi.org/10.1561/2200000083

[18] Anastasia Koloskova, Tao Lin, Sebastian U Stich, and Martin Jaggi. 2020. Decentralized Deep Learning with Arbitrary Communication Compression *(ICLR'20)*. https://openreview.net/forum?id=SkgGCkrKvH

[19] Anastasia Koloskova, Nicolas Loizou, Sadra Boreiri, Martin Jaggi, and Sebastian Stich. 2020. A Unified Theory of Decentralized SGD with Changing Topology and Local Updates *(ICML'20)*. https://proceedings.mlr.press/v119/koloskova20a.html

[20] Anastasia Koloskova, Sebastian Stich, and Martin Jaggi. 2019. Decentralized stochastic optimization and gossip algorithms with compressed communication *(ICML'19)*. https://proceedings.mlr.press/v97/koloskova19a.html

[21] Alex Krizhevsky, Vinod Nair, and Geoffrey Hinton. 2014. The CIFAR-10 dataset. 55, 5 (2014). https://www.cs.toronto.edu/~kriz/cifar.html

[22] Fan Lai, Yinwei Dai, Sanjay Singapuram, et al. 2022. FedScale: Benchmarking Model and System Performance of Federated Learning at Scale *(ICML'22)*. https://proceedings.mlr.press/v162/lai22a.html

[23] Xiangru Lian, Ce Zhang, Huan Zhang, Cho-Jui Hsieh, Wei Zhang, and Ji Liu. 2017. Can Decentralized Algorithms Outperform Centralized Algorithms? A Case Study for Decentralized Parallel Stochastic Gradient Descent *(NIPS'17)*. https://proceedings.neurips.cc/paper_files/paper/2017/file/f75526659f31040afeb61cb7133e4e6d-Paper.pdf

[24] Yujun Lin, Song Han, Huizi Mao, Yu Wang, and Bill Dally. 2018. Deep Gradient Compression: Reducing the Communication Bandwidth for Distributed Training *(ICLR'18)*. https://openreview.net/forum?id=SkhQHMW0W

[25] Yang Liu, Tao Fan, Tianjian Chen, Qian Xu, and Qiang Yang. 2021. FATE: An Industrial Grade Platform for Collaborative Learning With Data Protection. *J. Mach. Learn. Res.* 22, 226 (2021). http://jmlr.org/papers/v22/20-815.html

[26] Brendan McMahan, Eider Moore, Daniel Ramage, Seth Hampson, and Blaise Aguera y Arcas. 2017. Communication-efficient learning of deep networks from decentralized data *(AISTATS'17)*. https://proceedings.mlr.press/v54/mcmahan17a/mcmahan17a.pdf

[27] Message Passing Interface Forum. 2021. *MPI: A Message-Passing Interface Standard Version 4.0*. https://www.mpi-forum.org/docs/mpi-4.0/mpi40-report.pdf

[28] Christodoulos Pappas, Dimitris Chatzopoulos, Spyros Lalis, and Manolis Vavalis. 2021. IPLS: A Framework for Decentralized Federated Learning *(IFIP Networking'21)*. https://doi.org/10.23919/IFIPNetworking52078.2021.9472790

[29] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Kopf, Edward Yang, Zachary DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. 2019. PyTorch: An Imperative Style, High-Performance Deep Learning Library. In *NeurIPS'19*. https://proceedings.neurips.cc/paper_files/paper/2019/file/bdbca288fee7f92f2bfa9f7012727740-Paper.pdf

[30] Holger R Roth, Yan Cheng, Yuhong Wen, Isaac Yang, Ziyue Xu, YuanTing Hsieh, Kristopher Kersten, Ahmed Harouni, Can Zhao, Kevin Lu, Zhihong Zhang, Wenqi Li, Andriy Myronenko, Dong Yang, Sean Yang, Nicola Rieke, Abood Quraini, Chester Chen, Daguang Xu, Nic Ma, Prerna Dogra, Mona G Flores, and Andrew Feng. 2022. NVIDIA FLARE: Federated Learning from Simulation to Real-World. In *Workshop on Federated Learning: Recent Advances and New Challenges*. https://openreview.net/forum?id=hD9QaIQTL_f

[31] Rishi Sharma et al. 2022. decentralizepy: An open-source decentralized learning research framework. https://github.com/sacs-epfl/decentralizepy

[32] Nikko Strom. 2015. Scalable distributed DNN training using commodity GPU cloud computing. In *16th Annual Conference of the International Speech Communication Association (INTERSPEECH'15)*. https://www.isca-speech.org/archive_v0/interspeech_2015/papers/i15_1488.pdf

[33] Thijs Vogels, Hadrien Hendrikx, and Martin Jaggi. 2022. Beyond spectral gap: the role of the topology in decentralized learning *(NeurIPS'22)*. https://proceedings.neurips.cc/paper_files/paper/2022/file/61162d94822d468ee6e92803340f2040-Paper-Conference.pdf

[34] Thijs Vogels, Sai Praneeth Karimireddy, and Martin Jaggi. 2020. Practical Low-Rank Communication Compression in Decentralized Deep Learning *(NeurIPS'20)*. https://proceedings.neurips.cc/paper_files/paper/2020/file/a376802c0811f1b9088828288eb0d3f0-Paper.pdf

[35] Milos Vujasinovic. 2023. *Secure Aggregation on Sparse Models in Decentralized Learning Systems*. Master's thesis. EPFL. https://www.epfl.ch/labs/sacs/wp-content/uploads/2023/02/Secure_Aggregation_on_Sparse_Models_in_Decentralized_Learning_Systems___Milos_Vujasinovic.pdf

[36] Lin Xiao, Stephen Boyd, and Seung-Jean Kim. 2007. Distributed average consensus with least-mean-square deviation. *J. Parallel and Distrib. Comput.* 67, 1 (2007). https://doi.org/10.1016/j.jpdc.2006.08.010

[37] Timothy Yang, Galen Andrew, Hubert Eichner, Haicheng Sun, Wei Li, Nicholas Kong, Daniel Ramage, and Françoise Beaufays. 2018. Applied federated learning: Improving google keyboard query suggestions. (2018). arXiv:1812.02903

[38] Tongtian Zhu, Fengxiang He, Lan Zhang, Zhengyang Niu, Mingli Song, and Dacheng Tao. 2022. Topology-aware generalization of decentralized SGD *(ICML'22)*. https://proceedings.mlr.press/v162/zhu22d.html

[39] Alexander Ziller, Andrew Trask, Antonio Lopardo, et al. 2021. PySyft: A library for easy federated learning. In *Federated Learning Systems*. https://doi.org/10.1007/978-3-030-70604-3_5