FLOC: Federated Learning On a Cluster

Akshay Karkal Kamath*
Parima Devanshu Mehta*
ECE 6115 - Interconnection Networks
Georgia Institute of Technology

ABSTRACT

Privacy concerns have emerged in the era of Machine Learning as traditional approaches require training data to be sent to a server from edge devices (nodes). Moreover, the communication cost associated with sharing training data from nodes to the server is high. Federated Learning is emerging as a promising solution to conduct secure and efficient training of machine learning models. In the federated learning paradigm, training is carried out on individual edge devices in a collaborative manner and parameters are aggregated securely to form the global model. In this project, our objective is to demonstrate federated learning on a cluster of FPGA boards and conduct detailed performance analysis to understand the training times for centralized and decentralized instances of an off-the-shelf model. Moreover, we plan to deploy our federated model on the FPGA cluster to understand the networking overhead and its variation with scale. Based on our in-depth analysis, we aim to provide insights for future research directions in the field.

KEYWORDS

Federated Learning, On-device training, FPGA Cluster, Distributed Computation

1 INTRODUCTION

Machine Learning has transformed a plethora of industries such as automotive, health care, and finance, gaining ubiquity due to its ability to improve over time by making efficient use of data. In a traditional setting, training of machine learning models is achieved in a centralized fashion, wherein all the data from edge devices are moved to a computationally efficient server. However, this approach raises concerns pertaining to the privacy of users' sensitive data. Moreover, movement of a huge amount of training data to a server can be extremely slow and expensive. Federated Learning addresses these issues by enabling training of machine learning models in a secure and efficient fashion.

Federated learning aims at protecting the privacy of users' data by training a centralized model in a decentralized manner. To be more specific, training is moved to edge devices (nodes) rather than moving sensitive data to the server. To achieve this goal, the model is shared to all the edge devices from the server, where it is partially trained based on the data present on that device. Each node, instead of sharing data, sends the partially updated model to the server, where secure aggregation is performed to form a global model trained from all nodes. Several such iterations can be carried out to improve the accuracy of a federated model. This approach not only ensures privacy but also alleviates communication costs as transferring a model is cheaper than transferring training data.

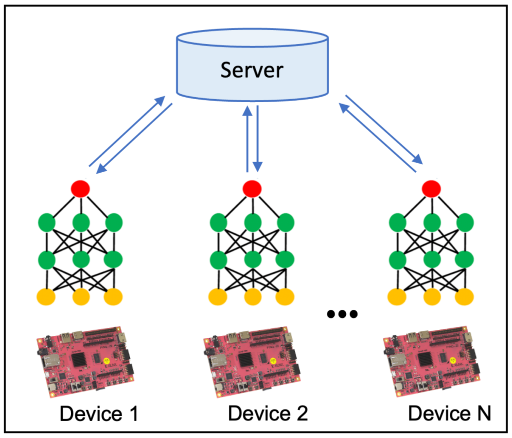

Figure 1: Top-level Diagram for Federated Learning on a FPGA Cluster

While the limitation of federated learning is that it requires edge devices to be capable of training machine learning models, the issue is not insurmountable and is expected to be solved soon [8].

In this project, we aim to demonstrate federated learning on a cluster of FPGAs, which is an area of research that remains highly unexplored [4]. First, we demonstrate distributed computation by setting up a network of Xilinx Pynq-Z2 boards that emulate edge devices. Second, we obtain the accuracy of a predictive model trained in a centralized manner and determine the number of iterations required by a federated model to match the same accuracy. The training time obtained from the simulation of the federated model will act as a baseline for further evaluation. Third, we deploy the federated model on the Processing Subsystem (PS) of all nodes in the cluster. Finally, we conduct performance analysis with respect to the baseline to understand the networking overhead as we increase the number of nodes in the cluster. Federated learning on a cluster is conceptually illustrated in Fig. 1

2 FPGA CLUSTER SETUP

To demonstrate federated learning on FPGAs, we employ the Pynq-Z2 board [12] which is a low-cost development board that features Xilinx Zynq SoC [17] and supports the PYNQ (Python Productivity for Zynq) framework [15]. Setting up of an individual Pynq-Z2 board is a well-documented process [16]. For Milestone 1, we built a cluster with three Pynq-Z2 boards and a Gigabit router as shown in Fig. 2.

For Milestone 2, we extend the cluster to support eight Pynq-Z2 boards using a Gigabit router, a network switch and a powered USB-hub. The final cluster setup is shown in Fig. 3.

*Both authors contributed equally to this research.

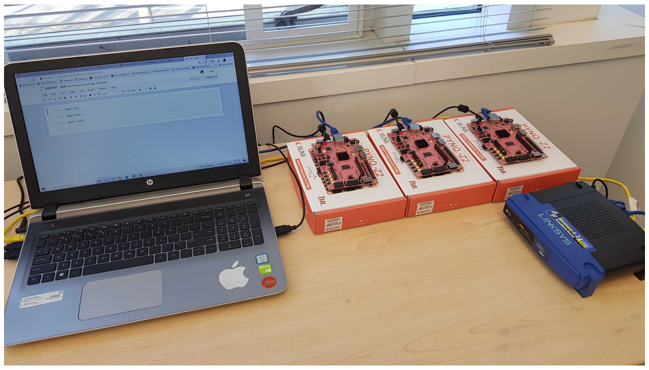

Figure 2: Initial cluster setup with 3 Pynq-Z2 boards connected to a Gigabit router.

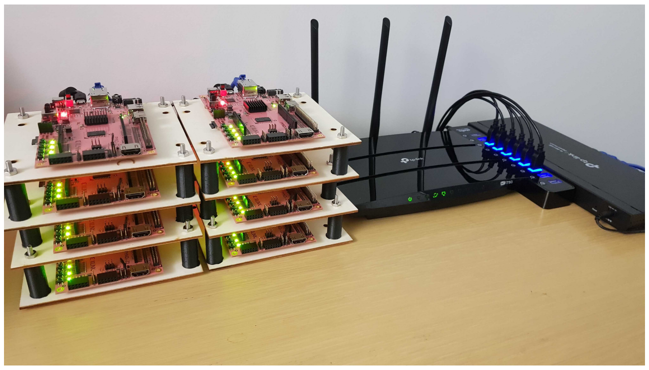

Figure 3: Final cluster setup with 8 Pynq-Z2 boards, Gigabit router, network switch, and powered USB-hub.

2.1 PYNQ Framework

PYNQ is an open-source project from Xilinx that enables developers to use Zynq devices (such as Pynq-Z2) without having to use ASIC-style design tools to design programmable logic circuits. PYNQ runs on top of the Processing System (PS) of the FPGA board (typically comprising of ARM cores) and allows configuration of the Programmable Logic (PL) with bitstream overlays exposed as Python libraries.

PYNQ devices can be accessed over USB using Serial Terminals or over a local area network using Ethernet [5]. The latter is a more preferable choice for two reasons:

- PYNQ adopts a web-based architecture which is browser agnostic and supports Jupyter Notebook framework served from the embedded processors (PS). This easy-to-use interface improves programmer's productivity.
- PYNQ also provides the provision for accessing serial terminals through the Jupyter Notebook, thereby acting as a super set to USB-based connection.

In our final cluster setup, each of the eight Pynq boards is connected to the same local area network. The IP addresses are dynamically assigned to the boards and are obtained from the router's settings webpage. The setup procedure is described in detail in Appendix A.

2.2 Dask.distributed Python Library

Dask.distributed [6] is a lightweight library for distributed computing in Python. It is a centrally managed, distributed, dynamic task scheduler. It provides low-latency peer-to-peer data sharing by removing central bottlenecks for data transfer. It also supports complex workflows required for sophisticated machine learning and data analytic applications while preserving data locality.

Dask.distributed comprises of three major components:

(1) **Scheduler**: Central process that is asynchronous and event driven, coordinating the computation requests from multiple clients and tracking the progress of multiple workers.
(2) **Worker**: Process that performs actual computation using a thread pool on a local (same as scheduler) or remote machine. Each FPGA is configured as a worker and connected to the central scheduler.
(3) **Client**: Python class that acts as the entry point to the Dask cluster. Users interact with the scheduler and workers through methods of this class.

2.3 Building Pynq-Z2 Cluster with Dask

The cluster architecture is shown in Fig. 4. Setting up of the FPGA cluster involves the following steps (a Linux laptop is used as the central server):

- Each Pynq-Z2 board is booted up and connected to a router over Ethernet.
- Dask.distributed library is installed on each of the three boards and the laptop.
- Dask Scheduler process is launched from a laptop terminal window. This generates an IP address for workers to connect to.
- Dask Worker process is run on each of the three FPGA boards. This binds the boards to the scheduler.
- A custom Python script with calls to Dask Client is run from another terminal session on the laptop. The setup is now ready to execute distributed computation tasks on the workers.

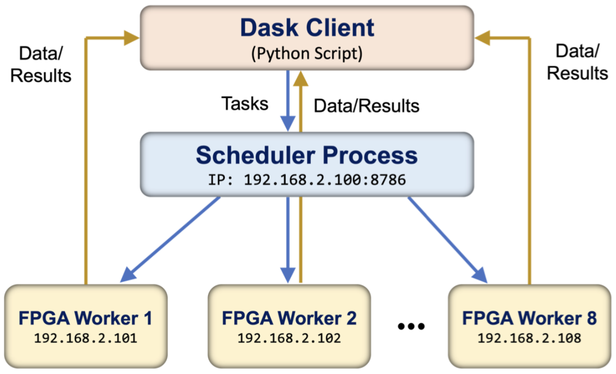

Figure 4: Cluster Architecture for Distributed Computation

Table 1: Results of Distributed Computation on Initial FPGA Cluster

Number of Workers	Execution Time (seconds)	Speedup
1	22.78	1.00x
2	11.92	1.91x
3	8.09	2.82x

3 CLUSTER SETUP EVALUATION

To demonstrate distributed computation, we write a program to compute the sum of the exponents of a randomly-initialized array with 100 million elements. The Dask library internally handles the distribution of array elements onto the individual workers. The client process terminates once the results are available from every connected board.

To obtain the baseline execution time, we first connect only one FPGA board (Worker 1) to the Scheduler and run the program. This provides us the sequential execution time of the application (as it is run on a single thread). Next, we connect Worker 2 and re-run the program. The task is now distributed equally between the two workers. Finally, we connect Worker 3 and run the program again. The execution time of each trial is shown in Table 1.

We repeat the same process of incrementally adding a new board to the cluster, running a Dask worker process on it and connecting it to the scheduler, and running the application to get the execution time in each case. The final cluster performance for up to eight FPGA boards is shown in Fig. 5.

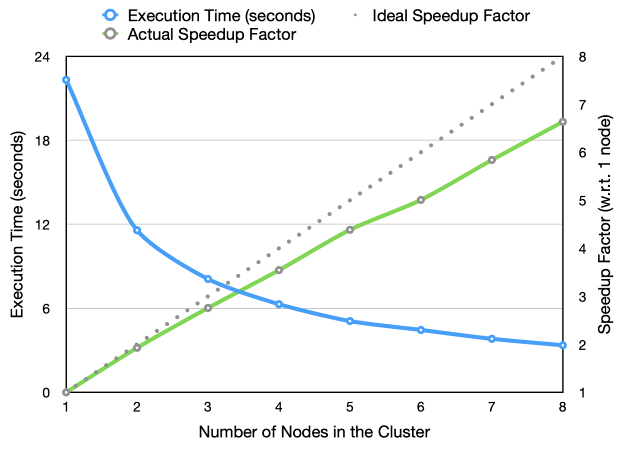

Figure 5: Cluster Performance Evaluation Results

With two workers, the execution time is almost half of the single-worker run time. Similarly, the execution with all three workers is nearly one third of the sequential execution time. However, as we keep adding a new node to the cluster, the actual speedup achieved deviates from the ideal speedup expected. For instance, with 8 nodes, the overall speedup achieved is only **6.65x** rather than 8x. The deviation from ideal execution times for distributed computation can be attributed to the overhead introduced by the network.

4 FEDERATED LEARNING PARADIGM

Most of the practical applications that leverage the potential of machine learning work with data generated by devices or services linked to humans who are the users. For instance, a smart watch monitors a person's heartbeat to detect abnormal activities, a smart keyboard learns from a user's writing patterns to make accurate predictions for next words, etc. To improve the performance of machine learning models, organizations typically resort to collecting such sensitive data from users in their cloud servers and use this centralized data for training their models.

To address the growing concerns over data privacy, researchers at Google introduced an alternate decentralized training approach called Federated Learning [10] that allows for learning of a shared prediction model collaboratively without requiring the users to share their data. The concepts of federated learning can be best understood from Google AI's comic on the same topic [7]. The efficacy of federated learning approach has been tested for Gboard on Android (the Google Keyboard) and demonstrated to allow for smarter models, lower latency, and less power consumption, all while ensuring privacy.

In the Federated Learning setting, data is distributed across end devices in a highly uneven fashion. Furthermore, these devices have significantly higher-latency, lower-throughput connections and may only be intermittently available for training. These bandwidth and latency limitations can be overcome through the Federated Averaging algorithm [9], which can train deep networks using 10-100x less communication compared to a naively federated version of stochastic gradient descent. In our project, we use the federated averaging algorithm to demonstrate federated learning on the FPGA cluster and explain it in detail in Section 5.2.

5 FEDERATED LEARNING SIMULATION

In this section, we simulate Federated Learning and compare its performance with a centralized model [14] to analyze the communication cost associated with decentralized, federated training. For evaluation, we implement a simple 2-layer classification model with parameters described in Table 2 and train it using the MNIST dataset which comprises of grayscale images (of size 28 x 28 pixels) of handwritten digits from 0 to 9.

For milestone 2, we modified an off-the-shelf PyTorch-based model for MNIST digit recognition [2]. However, as explained in Section 7, we had to re-write the model using only NumPy as Pynq-Z2 does not support PyTorch contrary to our initial assumption. We picked up a Numpy-based MNIST model [3] and extended the same to achieve federated learning.

Table 2: Parameters of the 2-layer Classification Model

Parameter	Value/Type
Input Layer	784
Hidden Layer	100
Output Layer	10
Activation Functions	ReLU, Softmax

5.1 Centralized Model

In a centralized model, the training process depicted in Fig. 6 comprises of the following steps:

- Training data from edge devices (nodes) is sent to a centralized server with high compute power.
- Model is trained on the entire training data for a certain number of epochs till its performance stabilizes.
- Post training, the updated model is sent back to the edge devices for on-device inference.

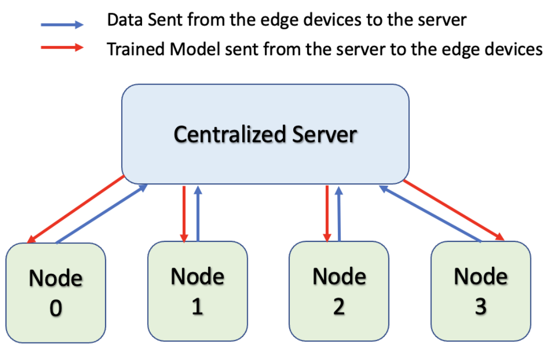
Data Sent from the edge devices to the server
Trained Model sent from the server to the edge devices
Centralized Server
Node 0
Node 1
Node 2
Node 3

Figure 6: Illustration of Centralized Training

5.2 Federated Model

In a federated model, training occurs in a decentralized manner [1] wherein each edge device (node) trains the model on a subset of training data individually.

To simulate decentralized training, we consider an ideal scenario where training data is evenly distributed across all nodes. To achieve this, we divide the MNIST dataset such that each node contains an Independent and Identically Distributed (IID) subset of the main dataset. We group the data based on their labels, shuffle their indices, and extract an equal amount of data per label, which is then evenly distributed to each node.

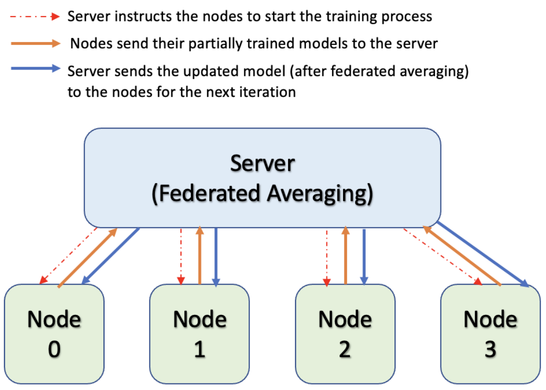
Server instructs the nodes to start the training process
Nodes send their partially trained models to the server
Server sends the updated model (after federated averaging) to the nodes for the next iteration
Server (Federated Averaging)
Node 0
Node 1
Node 2
Node 3

Figure 7: Illustration of Decentralized Training

As depicted in Fig. 7, decentralized training comprises of the following steps:

- The server sends an instruction to each node to begin the training process.
- Each node trains on a subset of its data separately for a certain number of epochs, and sends its partially trained model back to the server. Note that only the model parameters are sent back to the server, instead of the training data, thereby protecting data privacy.
- The server combines the models obtained from individual nodes to form a global model. Federated averaging involves merging the partially trained models by taking an average of their trained weights.
- The global model sends the updated model to the nodes, to start the next iteration. Several iterations may be required by the global model to match the performance of the centralized model.

5.3 Performance Comparison of Centralized and Decentralized Model

We conduct a comparative performance analysis of the centralized model and the federated model with 2, 4, and 8 nodes. For each federated model setting, we determine the number of federated rounds that it requires to achieve an approximately equal performance (test accuracy) as that of the centralized model. This allows us an insight into how the communication cost varies as the number of nodes are increased in a federated setting.

5.3.1 ***Evaluation Setup.*** Although the MNIST dataset consists of 50,000 training images and 10,000 test images, the number of data points for each label is not equal. Thus, we determine the label with the least number of data points and use the same number of data points for the remaining labels. To ensure a fair and consistent evaluation, we use the same setup to train centralized and decentralized models as described in Table 3.

Table 3: Training setup for Centralized and Decentralized Models

Configuration Parameter	Value
Training data	44,960
Test data	10,000
Training epochs	10
Batch size	100
Alpha	0.1

5.3.2 ***Centralized Model Training.*** We train the centralized model with model parameters described in Table 2 and training configuration mentioned in Table 3. The incremental model performance improvement after each epoch is shown in 8. The accuracy improves with the increase in number of epochs and a test accuracy of **96.26 %** is achieved after 10 epochs which acts as our baseline for federated model evaluation.

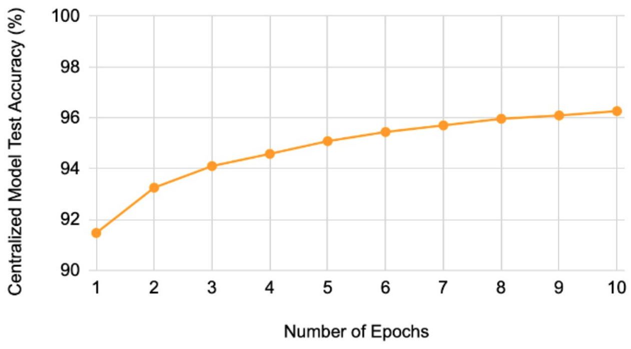

Figure 8: Baseline: Centralized Model test accuracy vs. epochs

5.3.3 ***Federated Model Training.*** We develop our model using *numpy* because of the following reasons:

- TensorFlow's federated learning framework called TensorFlow Federated (TFF) [13] cannot be installed on a memory-constrained device like Pynq-Z2.
- PyTorch cannot be installed on the Processing System of Pynq-Z2 device as it is incompatible with a 32-bit system.

Hence, we resort to *numpy* which is installed out-of-the-box on the Processing System (PS) of Pynq-Z2 board. We provide details of our code in Appendix B and C.

We train the federated model with training configuration mentioned in Table 3. To simplify our analysis, we consider an ideal scenario wherein training data is Identically and Independently Distributed (IID) per node as depicted in Table 4.

Table 4: Setup for Performance Evaluation of Federated Model against Centralized Model

Number of Nodes	Training Data per Node
2	22480
4	11240
8	5620

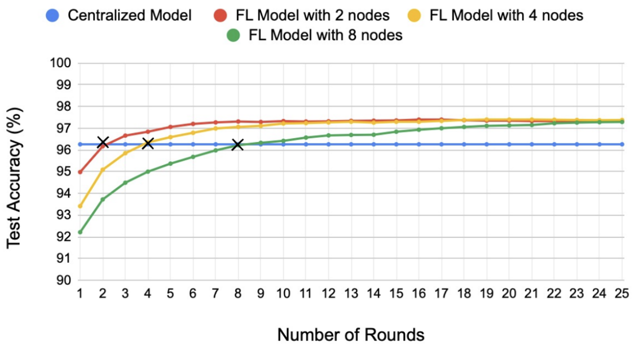

Figure 9: Performance Evaluation of Federated Model against Centralized Model

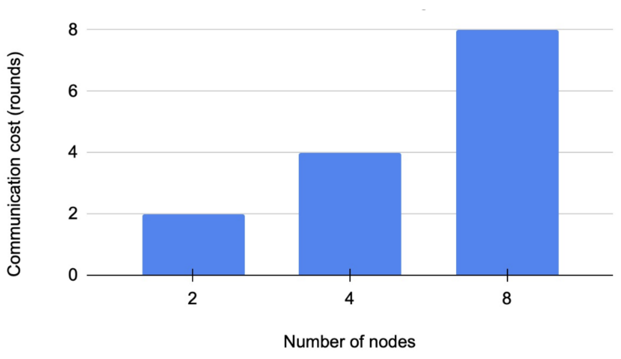

Figure 10: Communication cost associated with a Federated Model

Figure 9 shows that as the number of nodes increase in the federated model, the discrepancy between the performance of federated and centralized model increases, leading to a higher number of federated rounds to achieve the same test accuracy as that of the centralized model. We refer to the communication cost associated with the federated model as the "*Cost of Privacy*" as shown in Fig 10.

5.3.4 ***Analysis with Non-IID Data.*** In a practical scenario, the data on the individual nodes may not be IID owing to the nature of training data acquisition in the nodes. This is a fundamental problem that arises in a federated learning setting. To analyze how a random distribution of data across nodes impacts model performance, we created various subsets from the main MNIST dataset with an uneven data distribution:

- For dual-node cluster, one subset comprises of only the data points with labels between 0 and 4 (both inclusive) while the other subset contains the remaining data points with labels between 5 and 9 (both inclusive).
- For quad-node, a subset contains all the data points of pairs consecutive digits viz. 0-1, 2-3, 4-5, and 6-7, while the data points with labels 8 and 9 are equally distributed among all four subsets.
- For octa-node, a subset holds all the data points of a single digit from 0 to 7 while data points with labels 8 and 9 are equally distributed among all eight nodes.

We simulate federated learning for each of these uneven data distribution settings for 70 rounds and compare the overall model performance against the centralized baseline as shown in Fig. 11.

6 DEPLOYMENT ON PYNQ CLUSTER

To analyze the computation cost on hardware, we deploy our federated learning model on the homogeneous Pynq cluster in three different configurations. In the first configuration, we run the dask-worker process on two Pynq boards – Pynq-1 and Pynq-2 – and connect them to the dask-scheduler. We term this cluster configuration as dual-node. To ensure data localization, we split the MNIST training dataset into two equal subsets – dual_node_train0 and dual_node_train1 – and upload them on the Pynq-1 and Pynq-2 boards respectively.

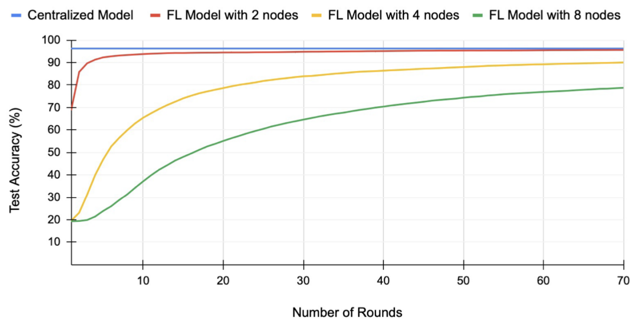

Figure 11: Federated Learning Analysis with Uneven Data Distribution

From a Dask client script, we then deploy a common model-training routine onto each of the two nodes using `client.submit()`. The Future objects immediately returned by the function calls are saved in separate variables and their `.result()` method is monitored in a blocking fashion. Once both the function calls return their results, averaging of model parameters is performed in the client script and the test accuracy of the model is determined. The process is repeated for the number of number of rounds obtained from simulation scripts and the hardware computation cost is analyzed for each federated round.

Similar process is followed for quad-node and octa-node cluster configurations. In each case, the model training is performed on the CPU cores (ARM Cortex A9) of the Pynq boards. Also, note that the same test dataset used for evaluating the baseline centralized model is utilized after each federated round for consistent comparison. The execution time per federated round for each configuration is demonstrated in Fig. 12

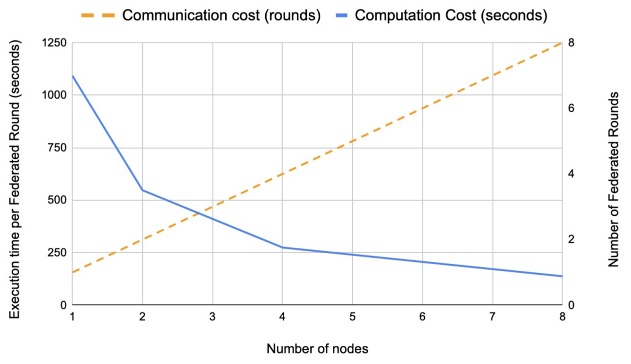

Figure 12: On-board Computation Cost vs. Communication Cost

For the same overall dataset, it can be observed that the computation burden reduces whereas the communication overhead increases as the number of nodes in the cluster increase.

7 SETUP ISSUES FACED

Several package dependency errors were encountered while setting up the Dask cluster. For instance, *msgpack* and *click* packages had to be reverted to older versions from the default ones to ensure workers could establish connections successfully with the scheduler. In addition, one of the tutorials on the official Dask website recommended the use of *jobqueue* package for deploying a Dask cluster. As it turns out, this was meant for HPC users, to be used alongside job queueing systems, different from what we were trying to accomplish. The current setup works in a streamlined manner without any of the queueing complications.

We had to completely re-write our MNIST digit recognition code post milestone 2 as we figured that PyTorch, similar to Tensorflow, is unsupported on Pynq-Z2. Our assumption was incorrectly based on the Ultra96v2 board which does support PyTorch. As it turns out, PyTorch library is written only for 64-bit architectures and while Ultra96v2 does have a 64-bit ARM Cortex M3 core, Pynq-Z2 doesn't as it has a 32-bit ARM core (Cortex-A9).

8 DESIGN CHOICES AND FUTURE WORK

We explain the rationale behind our design choices and describe possible directions to build on top of the setup that we have developed.

8.1 Pynq-Z2 over Raspberry Pi

While federated learning has been previously demonstrated on embedded processor platforms such as Raspberry Pi [11], its extension to FPGA boards is relatively unexplored. This motivated us to create an FPGA-based cluster infrastructure. Our next step will be to explore training of models on the programmable fabric of the FPGA boards and analyze the trade-offs of energy-efficient acceleration with fixed-point precision against the model's accuracy.

8.2 Realistic Cluster Setup

In this project, we employed a cluster with homogeneous nodes with equal computation and network capacities to demonstrate a proof-of-concept working of federated learning. However, in a practical scenario, the nodes are heterogeneous with varied computation and network capabilities. Going forward, we will extend the setup to include different FPGA boards such as Ultra96v2, VC707, etc. and analyze their impact on computation and communication costs.

8.3 Performance Evaluation

We assumed a top-down approach to analyze the performance of a model in a centralized setting and compare it with a decentralized implementation in this project. However, in a practical scenario, centralized model baseline cannot be determined as the data is localized to the nodes. This requires a bottom-up evaluation where the size of the dataset is not constant, but instead increases with each new node added to the cluster. A plausible approach would be to train a centralized model by merging the datasets present on connected nodes and then estimating the baseline for comparison.

9 CONCLUSION

We split our project into three phases and achieved our goals successfully. For milestone 1, we built a cluster with three nodes and demonstrated distributed computation on it. For milestone 2, we simulated federated learning for 2, 4, and 8 clients using a 2-layer classifier for MNIST dataset. Further, we evaluated the performance of the federated model against a centralized model and observed that the number of iterations required by the federated model to converge increases with the number of nodes (edge devices). Finally, we deployed the federated model on the cluster setup to quantitatively analyse the hardware cost for federated learning on the edge.

To summarize, we successfully demonstrated a plain vanilla federated learning model implementation on a homogeneous cluster of FPGA boards and analyzed the communication and computation cost for an off-the-shelf model with three different cluster configurations. We hope to explore and build on the algorithmic, networking and hardware components to develop a comprehensive framework for federated learning in multi-FPGA setting.

10 ACKNOWLEDGEMENT

The authors would like to thank Prof. Cong (Callie) Hao, Assistant Professor, Department of Electrical and Computer Engineering, Georgia Institute of Technology, for providing access to the Pynq-Z2 cluster at the Software-Hardware Co-Design Lab (Sharc Lab).

REFERENCES

[1] 2018. Federated Learning. (November 2018). https://federated.fastforwardlabs.com.

[2] 2020. Federated Learning: A Simple Implementation of FedAvg (Federated Averaging) with PyTorch. (September 2020). https://towardsdatascience.com/federated-learning-a-simple-implementation-of-fedavg-federated-averaging-with-pytorch-90187c9c9577.

[3] 2022. MNIST from Scratch. (January 2022). https://python.plainenglish.io/mnist-from-scratch-3335ca7b309b.

[4] Haftay Gebreslasie Abreha, Mohammad Hayajneh, and Mohamed Adel Serhani. 2022. Federated Learning in Edge Computing: A Systematic Survey. *Sensors* 22, 2 (2022), 450.

[5] Shashank Aggarwal. 2020. *Scaling up data analytics in Python using multiple FPGAs.* Master's thesis. Delft University of Technology.

[6] Anaconda, Inc. 2022. *Dask.distributed.* Anaconda, Inc. https://distributed.dask.org/en/stable/.

[7] Lucy Bellwood and Scott McCloud. 2017. *Federated Learning: An online comic by Google AI.* https://federated.withgoogle.com.

[8] Dongqi Cai, Qipeng Wang, Yuanqiang Liu, Yunxin Liu, Shangguang Wang, and Mengwei Xu. 2021. Towards Ubiquitous Learning: A First Measurement of On-Device Training Performance. In *Proceedings of the 5th International Workshop on Embedded and Mobile Deep Learning* (Virtual, WI, USA) *(EMDL'21)*. Association for Computing Machinery, New York, NY, USA, 31–36. https://doi.org/10.1145/3469116.3470009

[9] Brendan McMahan, Eider Moore, Daniel Ramage, Seth Hampson, and Blaise Aguera y Arcas. 2017. Communication-Efficient Learning of Deep Networks from Decentralized Data. In *Proceedings of the 20th International Conference on Artificial Intelligence and Statistics (Proceedings of Machine Learning Research, Vol. 54)*, Aarti Singh and Jerry Zhu (Eds.). PMLR, 1273–1282. https://proceedings.mlr.press/v54/mcmahan17a.html

[10] Brendan McMahan and Daniel Ramage. 2017. *Federated Learning: Collaborative Machine Learning without Centralized Training Data.* https://ai.googleblog.com/2017/04/federated-learning-collaborative.html.

[11] Rini Apriyanti Purba and Wen Hao. 2021. *Implementation of Federated Learning on Raspberry Pi Boards.* Master's thesis. KTH ROYAL INSTITUTE OF TECHNOLOGY.

[12] Technology Unlimited (TUL) 2021. *Pynq-Z2 Development Board.* Technology Unlimited (TUL). https://www.tulembedded.com/FPGA/ProductsPYNQ-Z2.html.

[13] TensorFlow 2022. *TensorFlow Federated.* TensorFlow. https://www.tensorflow.org/federated.

[14] Lorenzo Varallo. 2022. MNIST from Scratch: An attempt at understanding Neural Networks more deeply. (January 2022). https://medium.com/python-in-plain-english/mnist-from-scratch-3335ca7b309b.

[15] Xilinx 2021. *PYNQ: Python productivity for Xilinx platforms.* Xilinx. https://pynq.readthedocs.io/en/latest/index.html.

[16] Xilinx 2021. *Pynq-Z2 Setup Guide.* Xilinx. https://pynq.readthedocs.io/en/latest/getting_started/pynq_z2_setup.html.

[17] Xilinx 2022. *Zynq-7000 SoC.* Xilinx. https://www.xilinx.com/products/silicon-devices/soc/zynq-7000.html.

A FPGA CLUSTER SETUP PROCEDURE

A.1 Pynq-Z2 Board Setup

The process to get a PYNQ-Z2 board up and running is rather straightforward. The latest SD card boot image can be downloaded here. The boot image can be written onto the SD xard by following the steps described in this link. Finally, the steps to boot the board can be found here.

After sliding the power switch on the board on the ON position,

- The **Red** LED will comes on immediately to confirm that the board has power.
- After a few seconds, the **Yellow/Green/Done** LED light up to show that the device is operational.
- After a minute, two **Blue** LEDs and four **Yellow/Green** LEDs flash simultaneously.
- The **Blue** LEDs then turn on and off while **Yellow/Green** LEDs remain on.

The system is now booted and ready for use.

A.2 Connecting To The Local Area Network

To use the Jupyter Notebook enabled by default on Pynq, the board has to be connected to a network over Ethernet. We use a Gigabit router and a network switch to connect all the PYNQ-Z2 boards to the same network. The dynamic IP address of each board can be obtained from the router's settings page. The corresponding Jupyer Notebooks can be accessed by entering the IP addresses in different tabs of a web browser.

A.3 One-Time Board Configuration

When working with multiple Pynq boards connected over the same network, it is a good idea to assign a unique identifier to each board. This can be done as follows:

- Open a terminal window in Jupyter (New -> Terminal)
- Run the script `pynq_hostname.sh <board_name>`
- Reboot the board and connect to it as described in Appendix A.2

A.4 Installing and Configuring Dask

To ensure correct connection between the Dask scheduler and any Dask worker, the Dask package versions need to be same across all entities. The steps to correctly install Dask on PYNQ-Z2 are as follows:

- In a Jupyter terminal, run
  ```
  python -m pip install "dask[distributed]"
  ```
- Downgrade **click** package to version 7.1.2 using
  ```
  pip install click==7.1.2
  ```

- As the latest version (1.0.3) of **msgpack** has some deserialization issue, downgrade the same by running `pip install msgpack==0.6`.

A.5 Testing Dask Cluster

To test the Dask cluster with variable number of nodes (FPGA boards), we run a sample client script that runs a sample distributed computation application on the number of nodes present. This script is available on our GitHub repository here. The steps involve,

- Run `dask-scheduler` in a laptop terminal (needs to be run only once)
- On each Pynq board connected, run an instance of dask worker on it using `dask-worker <scheduler_addr>:8786 –name <name>`.
- The worker is successfully connected to the scheduler when the Jupyter terminal shows the message `Starting Established Connection`.
- In another terminal window on the laptop, run the client script to check worker connection and the results of distributed computation.

B FEDERATED LEARNING SIMULATION

The links to Jupyter Notebooks for various federated learning experiment settings are listed below.

- Federated learning with 2 nodes.
- Federated learning with 4 nodes.
- Federated learning with 8 nodes.

C ON-BOARD DEPLOYMENT

Each board must have a copy of `fl.py` which gets invoked while deploying the training routing. In addition, depending on the cluster configuration, a node must include the relevant datasets for training the model locally with their respective data.