Emulating Federated Learning on an FPGA Cluster

Abhipsa Panigrahi Devansh Johri Khatib Mohammed Adeeb Virat Agarwal

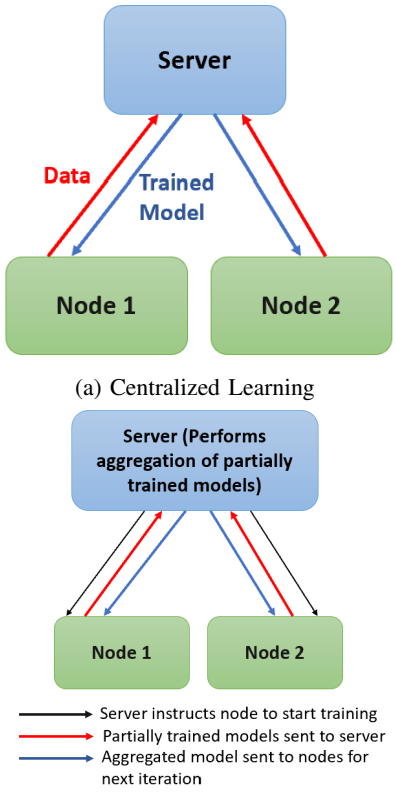
***Abstract*—Traditional Machine Learning requires sending all data to a central computationally-efficient server where the entire model is trained. The concerns associated with this approach are primarily data privacy, the latency of data movement, and the energy associated with the same. In Federated Learning, a Machine Learning algorithm is trained across multiple decentralized edge devices holding local data samples, followed by secure aggregation of parameters on the central server. It thus overcomes the above concerns associated with the former approach. In this report, we introduce the concept of Federated Learning in detail and describe its implementation in an FPGA cluster. Our objective in this project is to demonstrate Federated Learning on a cluster of FPGA boards.**

***Index Terms*—Federated Learning, FPGA cluster, data privacy, latency, edge devices**

I. Introduction

Traditionally, Machine Learning model training is carried out in a centralized manner, where the edge devices (nodes) need to send all their data to a central server. This server is computationally very efficient. The entire model is trained here using all of the data, and then the trained model is sent back to the nodes for local inference. This centralized training method raises a major concern about data privacy since all data leaves the nodes and has to go to the server. Moreover, the latency associated with the movement of such huge volumes of training data is very large, and so is the energy cost associated with it. Federated Learning addresses these issues through a distributed and secure training method.

Federated Learning essentially means training a centralized model in a decentralized manner. In this paradigm, instead of moving all data from nodes to a central server, the training of the model is distributed between individual nodes. Let us now understand the steps in how this is done. The server first sends the model to all the edge devices (nodes). The model is partially trained on each node using the local data present on that particular node. So, data stays where it is, and does its part in partial training of the overall model. All the nodes then send the partially updated models to the central server, where secure aggregation of the model parameters is performed to form a global model trained with data from all the nodes. This completes a single iteration of training. For another successive iteration, the server sends back the aggregated model to all the nodes. The nodes again perform partial updation of the model parameters based on their local data. Several such iterations are done in order to train the overall model to achieve good accuracy. Hence, Federated Learning not only preserves data privacy but also reduces communication overhead because transferring model parameters is much cheaper than transferring entire training data.



(a) Centralized Learning

(b) Federated Learning

Fig. 1: Centralized Learning vs Federated Learning

In the course of this project, our objective is to demonstrate Federated Learning on a cluster of Xilinx Pynq-Z2 FPGA boards (acting as nodes). The host device (server) can either be a host PC (e.g. raspberry pi) or another Pynq-Z2 FPGA board. Since complete training of a model would require very high computation power and resources, we aim to begin by demonstrating Transfer Learning in a distributed manner. Transfer learning is a machine learning technique where a model trained on one task is used as a starting point for training a new model on a different but related task. Since the pre-trained model has already learned a set of features that are relevant to the new task, it significantly reduces the amount of data, power, and resources required to train it for the new task. Our aim is to demonstrate federated transfer learning to retrain only the last fully-connected layer of an off-the-shelf neural network model.

II. Problem Description

Machine Learning has come to be widely used in numerous domains for many different tasks. Data is the backbone of all machine learning models. The more and better representative the data that the model is trained on, the better it will perform on new unseen data. Traditional model training is performed on a centralized server that is computationally very powerful. It requires that all the training data should be transferred from all sources of collection to the centralized server. This entails severe data privacy and security concerns. The data might often contain sensitive information such as patient health records or defense information, and if the central repository is hacked, the consequences would be severe. Moreover, the movement of huge volumes of training data from their sources to the central server has huge latency and energy costs associated with it. It might often be challenging due to network and bandwidth limitations.

Federated learning is a technique that addresses all these issues by allowing the training of machine learning models in a decentralized manner without centralizing the data. The data remains on the devices or in the local servers where it was collected (referred to as nodes). Training is instead brought onto the nodes. Federated learning enables multiple nodes to collaborate on model training without exchanging raw data. In this setting, data is distributed across nodes unevenly. Furthermore, these devices may only be intermittently available for training and have many higher-latency connections. These limitations can be overcome by using the Federated Averaging Algorithm.

This approach has several advantages. First, it preserves the privacy of the data, as it never leaves the node. Second, it enables training on data distributed across multiple nodes without requiring the data to be transferred to a central location, which reduces network and bandwidth requirements substantially. Only the model parameters need to be transferred back and forth between the nodes and the central server, which is much cheaper in terms of latency and bandwidth costs than transferring entire training datasets. These advantages make Federated Learning a promising alternative to traditional learning methods used in Machine Learning.

The limitation associated with Federated Learning is that edge devices (nodes) need to have training capability. It is a well-known fact that training of large Machine Learning models requires a huge amount of computational resources. Not all edge devices have this ability. In this project, we aim to use Xilinx Pynq-Z2 FPGA boards as edge devices for Federated Learning. The primary reasons for choosing FPGA for this purpose are hardware flexibility, low power consumption, and efficient resource utilization. We intend to use the programmable fabric of the FPGA to train an off-the-shelf machine learning model's fully-connected layer using the Transfer Learning method. The choice of device for acting as the central server is yet to be made.

III. Related Work

Federated Learning has already been proposed and used in the context of medical applications [1], vehicle-to-vehicle communication [2], etc. Google has used Federated learning for a predictive mobile keyboard that enables next-word prediction without sending private user data to the server [3].

Wang et al. [4] proposed a hardware/software (HW/SW) co-designed FPGA accelerator for federated learning. In the HW part, they introduced a Hardware-aware Montgomery Algorithm which utilized data parallelism and pipeline, and designed an FPGA architecture to decouple data access and computation. Along similar lines, Yang et al. [5] also designed an FPGA-based framework to accelerate the training phase in federated learning. As per the survey published by Abreha et al. [6], FPGAs are a very attractive choice for training Machine Learning models on edge due to their high energy efficiency.

IV. Proposed Approach

We will focus on the two pillars of Federated Learning -

A. Learning

Each FPGA in the system will be able to independently run the Machine Learning Algorithm on its training set as part of its Programmable Logic (PL). To train the input dataset, each PL will pass its input via a Convolutional Layer followed by a Fully Connected Layer (Fig 2). For our implementation, we will employ Transfer Learning where we will keep the weights of the Convolutional Layer and all other intermediate layers constant, and only update the parameters of the last Fully Connected Layer.

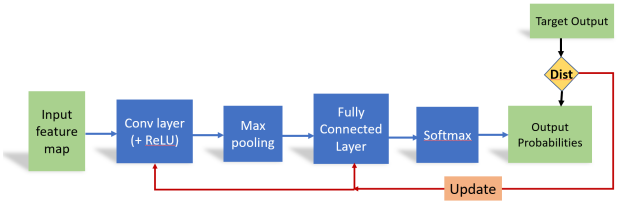


Fig. 2: Complete Training Model

Our learning process can be broadly classified into two steps: the forward pass and the backward pass.

The forward pass, also known as inference or prediction, involves passing the input data through a trained model to obtain a prediction or output. During the forward pass, the data flows through the different layers of the model, and each layer performs a set of mathematical operations on the input to produce an output. The output of one layer serves as the input to the next layer until the final output is produced.

The backward pass, also known as backpropagation, involves calculating the gradients of the loss function with respect to the parameters of the model. The gradients are then used to update the model parameters through an optimization algorithm, such as stochastic gradient descent (SGD), to minimize the loss function. During the backward pass, the

gradients are calculated by propagating the error from the output layer to the input layer, and the chain rule of calculus is used to calculate the partial derivatives of the loss function with respect to each parameter in the model.

B. Federated Transactions

With the model trained on individual FPGAs and the weights (for the fully connected layer) computed, the Federated Averaging PS (Federated PS) will collect the weights, average them and send the average weights to all the devices. Fig-3 captures the connections between the FPGA Devices and the Federated PS.

For our implementation, the Federated Averaging PS will poll across each device and check if their respective weight matrices are ready. If an FPGA is ready with its weights, the Federated PS will trigger the respective PS to send the weights over the Ethernet channel. Once the collection from all the devices has been done, the Federated PS will average the weights to create a single weight. The averaged weight will then be broadcast to all the FPGAs via their respective ethernet channels and the PS of each device will be instructed to trigger their PL to start the second training epoch with the averaged weight. The PS of each device will forward the average weight to their respective PL via the Master AXI bus and trigger the PL. Each PL will again do the training with the average weight as a starting point and create a new weight. This process will run iteratively until the model converges.

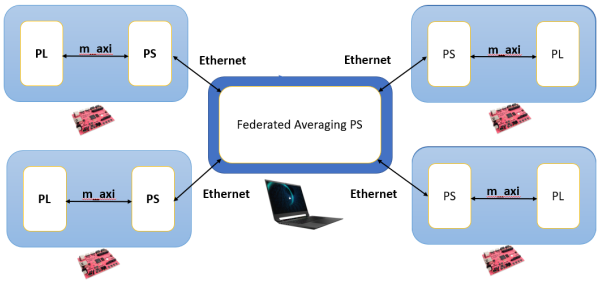


Fig. 3: System Connections

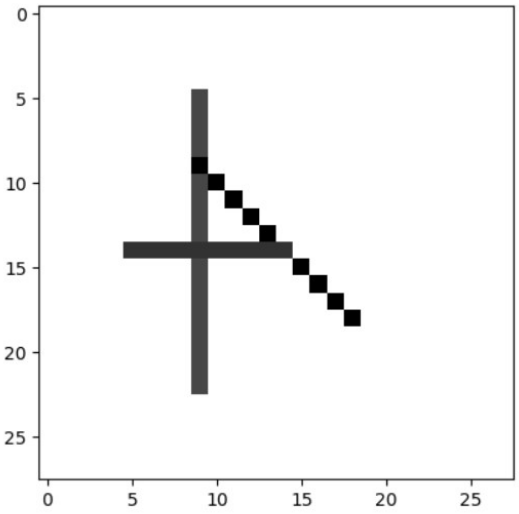


Fig. 4: Pytorch output for a sample image

B. Inferring the Pytorch-based model in FPGA

The inference process or the forward pass involves using a kernel to process image data with the detailed flow as captured below.

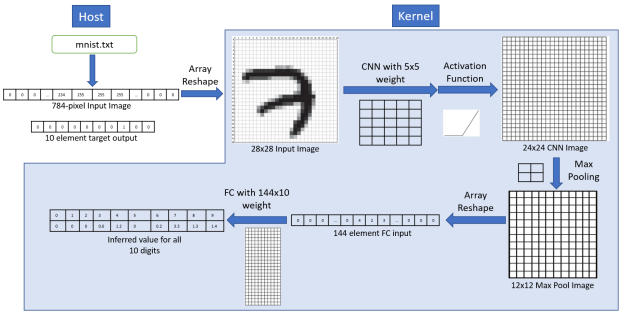


Fig. 5: Forward Pass - Model Structure

V. Mid-term Milestones

A. Generating the initial weights for the Machine Learning Algorithm

We trained a Pytorch-based model on a subset of the MNIST dataset for 50 epochs. We received and saved the Convolutional Layer and Fully Connected Layer weights. A sample output of the inference drawn on the server is captured in Fig. 4. We then used these trained parameters to draw inferences in simulation using our C++ kernel code. The plan is to send the parameters to the PL of the FPGA and draw inferences there which will be the forward-pass part of our training of the fully connected layer.

1) Each input image data is a flattened array of size 784, representing 784-pixel values of a 28x28 image.
2) A convolution operation is performed using a 5x5 kernel to transform the 28x28 image into a 24x24 convoluted image.
3) The resulting output is passed through a ReLU activation function. MaxPooling operation is then applied using a 2x2 filter to reduce the image size to 12x12.
4) The 12x12 2D matrix is converted into a 1D matrix of size 144 and passed through a linear layer with a size of 144x10.
5) The 10 output categories represent the probabilities of the 10 digits.

Finally, the element with the maximum output probability represents the predicted number.

```
[vagarwal307@ece-linlabsrv01 Final_Project]$ ./csim.out
Beginning Inference...
Processing Tile 1/16
Processing Tile 2/16
Processing Tile 3/16
Processing Tile 4/16
Processing Tile 5/16
Processing Tile 6/16
Processing Tile 7/16
Processing Tile 8/16
Processing Tile 9/16
Processing Tile 10/16
Processing Tile 11/16
Processing Tile 12/16
Processing Tile 13/16
Processing Tile 14/16
Processing Tile 15/16
Processing Tile 16/16

Output feature map:
Inferred value : 0
Inferred value : 0
Inferred value : 0
Inferred value : 0.593384
Inferred value : 1.19604
Inferred value : 0
Inferred value : 0.160156
Inferred value : 3.31714
Inferred value : 1.34595
Inferred value : 1.4281
 Inference completed!
```

Fig. 6: Snapshot of Inference Simulation

C. Federated Averaging in the FPGA

Initially, we used a setup where a server sends and receives data from a single PynqZ2 board. We are using socket programming to achieve federated communication between the server and PS section of the FGPA. For this, a unique IP address is assigned to each FPGA. Also, for every server-PS connection establishment, a new port is assigned. For basic functionality testing purposes, we made two 2-D integer numpy arrays with dimensions 100x150 and 150x200.

Working: The server contains a list of all the IP addresses and loops over each of them. In the first step, a connection is established with an FPGA board and the two arrays are sent to it after converting them to bytes. If the size of the array to be sent is large (larger than the receiver buffer) then the array is split before sending. Fig. 7 highlights the array sent by the server. On the receiver side, the FPGA receives the array of bytes from the buffer, which is then converted to int type which is captured in Fig. 8. We then did a matrix multiplication both on the PS and PL side to compare the computed results. In the second step, the server again loops over all the IP addresses and establishes a new connection (on a different port). The connected FPGA sends its computed array back to the server in a similar manner, after converting it to bytes. The server finally receives the data from its receiver buffer and displays the computed data which is captured in Fig. 9.

This helps us conclude that the connection cycle has been effectively established on one FPGA, and our next step is to test the same on multiple FPGAs and integrate the system on multiple FPGAs physically.

```
Connected by ('127.0.0.1', 49785)

1st Array sent:  [[399   2  29 ... 262 303 995]
 [495 564 396 ... 304 792 798]
 [758  88 549 ... 850 709 921]
 ...
 [493 781 521 ... 739 914 594]
 [510 542 241 ... 187 786 749]
 [770 596 565 ... 116 729 506]]

2nd Array sent:  [[775 526 836 ... 553 589 526]
 [493 428 801 ... 286 350 290]
 [310 415 142 ... 926 145 178]
 ...
 [670 287 568 ... 609 236 541]
 [848 784 298 ... 641 907 542]
 [424 958 917 ... 379 420 996]]
```

Fig. 7: Snapshot of array sent from the server

```
1st Array received:  [[399   2  29 ... 262 303 995]
 [495 564 396 ... 304 792 798]
 [758  88 549 ... 850 709 921]
 ...
 [493 781 521 ... 739 914 594]
 [510 542 241 ... 187 786 749]
 [770 596 565 ... 116 729 506]]

2nd Array received:  [[775 526 836 ... 553 589 526]
 [493 428 801 ... 286 350 290]
 [310 415 142 ... 926 145 178]
 ...
 [670 287 568 ... 609 236 541]
 [848 784 298 ... 641 907 542]
 [424 958 917 ... 379 420 996]]
```

Fig. 8: Snapshot of array received by the FPGA

```
Connected by ('127.0.0.1', 49786)

Array received:  [[ 20218   3505  14945 ... -31042 -26183  28994]
 [ 14437 -17856 -23266 ...  16060   9810  24693]
 [-15277  11402 -10181 ...  21846  32533  30544]
 ...
 [ -1169 -16782  -5780 ...  -9965 -20834  12363]
 [-10768 -15957   7459 ...  -2365  10955    -33]
 [-25194 -11285  20478 ... -13156   1823  32752]]
```

Fig. 9: Snapshot of array received by the server

VI. Milestones achieved post-Mid-term

A. Training the Fully Connected Layer in the FPGA

Using the initial weights obtained from training the PyTorch-based model for 50 epochs, we obtain the trained weights of the convolutional and Fully Connected layers. This weight becomes the input for the training of the Fully Connected Layer. Training of the fully connected Layer has been achieved via SGD by following these steps -

1) Forward Pass: Input data would be multiplied with the weights of the fully connected layer which are obtained after training the PyTorch model for 50 epochs.

2) Compute Loss: The loss between the output of the fully connected layer and the true labels is then computed using a suitable loss function.
3) Backward Pass: We then calculated the gradients of the loss with respect to the output of the fully connected layer. To obtain the predicted probabilities associated with each digit, a softmax activation function was applied to this output. After obtaining the probability values, the expected y labels were obtained and converted into a one-hot encoded vector format. The predicted probabilities and the expected one-hot encoded vector were used to calculate the loss values. To perform backpropagation, the stochastic gradient descent approach was utilized, as the fully connected layer parameters were updated after each training image. Firstly, the loss was calculated, and then the derivative of the loss with respect to the fully connected layer parameters was determined using the chain rule. The fully connected layer parameters were updated based on the gradient, which can be found in the equation in Fig. 10. for updating FC layer parameters. These gradients were then be used to compute the gradients of the loss with respect to the weights of the fully connected layer using the chain rule.

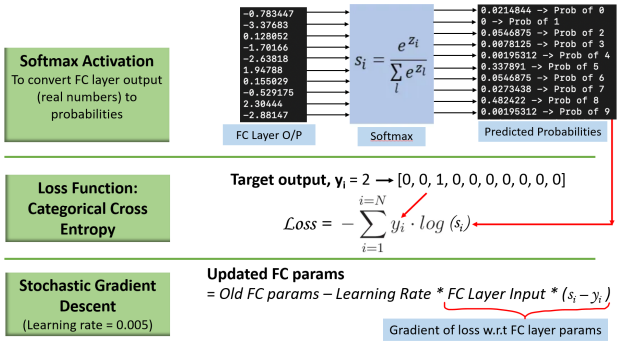


Fig. 10: Steps of the backpropagation process

4) Update Weights: Finally, the weights of the fully connected layer were updated using the computed gradients. These updated weights were be sent to the server. The averaged weights received from the server were used in the next iteration of training.
5) The kernel then iterates over Steps 1 through 4 for multiple epochs until the model has learned to make accurate predictions.

The results of the training with image1 and image2 as captured in Fig. 11, demonstrate a gradual decrease in Fully connected parameters from the first to the second image. This observation serves as a validation of our understanding of the backpropagation process.

B. Multiple FPGA Communication

We were able to extend our socket programming code to multiple FPGAs to enable communication between them. We

Fig. 11: Change in FC parameters between images

assigned unique IP addresses to each FPGA and established a connection cycle between them using TCP socket programming (as already implemented). We did test the system with larger data sets and ensured that communication between the FPGAs and the Federated Averaging Server is robust and reliable with results the same as captured in Fig. 12. The multiple FPGA communication enhances the overall system performance and help us accomplish decentralized or federated communication.

C. Combining Everything

Once ensured that the Server and Device PS are able to communicate perfectly and the PLs are able to generate the weights efficiently, we combined the two and trained our ML model on a few images from the MNIST training data set. As per our expectation, we were able to observe different weights coming from each of the FPGA devices (since they are all training the model on different data) to the Federated Averaging Server, and based on the categorical loss entropy we were able to observe a reduction in loss between the epochs as seen in Fig. 13.

```
Connected by ('192.168.0.101', 33610)
1st Array sent:  [990 682 390 ... 133 831 527]
2nd Array sent:  [ 76 304 743 ... 355 640 224]
Connected by ('192.168.0.102', 42088)
1st Array sent:  [990 682 390 ... 133 831 527]
2nd Array sent:  [ 76 304 743 ... 355 640 224]
Connected by ('192.168.0.103', 39964)
1st Array sent:  [990 682 390 ... 133 831 527]
2nd Array sent:  [ 76 304 743 ... 355 640 224]
Connected by ('192.168.0.104', 59118)
1st Array sent:  [990 682 390 ... 133 831 527]
2nd Array sent:  [ 76 304 743 ... 355 640 224]

Connected by ('192.168.0.101', 46054)
Array received from FPGA- 1 :  [[-12746   6688  29355 ...   6315 -26454 -30150]
 [ 25718   2066   2373 ... -32266 -28231   4934]
 [ -2632  -1016 -10819 ... -31653 -21081 -12432]
 ...
 [-14413  -8449  -5396 ... -18489  17196  29734]
 [-28915  24995  28487 ...  22857   8438  23555]
 [ 14940 -17889  30550 ... -11491  14565   3814]]
Connected by ('192.168.0.102', 49856)
Array received from FPGA- 2 :  [[-12746   6688  29355 ...   6315 -26454 -30150]
 [ 25718   2066   2373 ... -32266 -28231   4934]
 [ -2632  -1016 -10819 ... -31653 -21081 -12432]
 ...
 [-14413  -8449  -5396 ... -18489  17196  29734]
 [-28915  24995  28487 ...  22857   8438  23555]
 [ 14940 -17889  30550 ... -11491  14565   3814]]
Connected by ('192.168.0.103', 54826)
Array received from FPGA- 3 :  [[-12746   6688  29355 ...   6315 -26454 -30150]
 [ 25718   2066   2373 ... -32266 -28231   4934]
 [ -2632  -1016 -10819 ... -31653 -21081 -12432]
 ...
 [-14413  -8449  -5396 ... -18489  17196  29734]
 [-28915  24995  28487 ...  22857   8438  23555]
 [ 14940 -17889  30550 ... -11491  14565   3814]]
Connected by ('192.168.0.104', 40470)
Array received from FPGA- 4 :  [[-12746   6688  29355 ...   6315 -26454 -30150]
 [ 25718   2066   2373 ... -32266 -28231   4934]
 [ -2632  -1016 -10819 ... -31653 -21081 -12432]
 ...
 [-14413  -8449  -5396 ... -18489  17196  29734]
 [-28915  24995  28487 ...  22857   8438  23555]
 [ 14940 -17889  30550 ... -11491  14565   3814]]
```

Fig. 12: Establish connection and send/receive data from multiple FPGAs

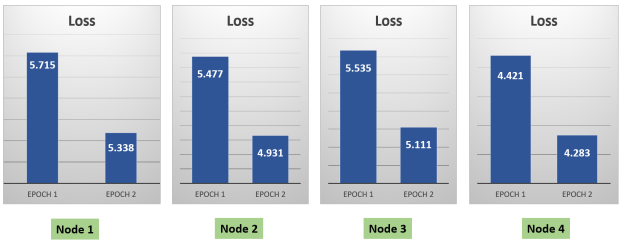


Fig. 13: Loss reduction between Epochs

D. Design Space Exploration

After thorough testing of the above design implementation, we extended the same to:

- Observe the relationship of the performance as a function of the number of FPGA devices connected to the Federated Average PS.
- Study the compute-communication trade-offs of the approach.

VII. Conclusion

In this project, we successfully demonstrated the feasibility of emulating Federated Learning on an FPGA Cluster. Our results show that FPGA clusters offer a promising approach for scaling Federated Learning to large-scale distributed systems, where communication and computation bottlenecks can often limit performance. Moreover, the flexibility and programmability of FPGAs make them well-suited for customizing and optimizing machine learning workloads, allowing for even greater efficiency gains in the future.

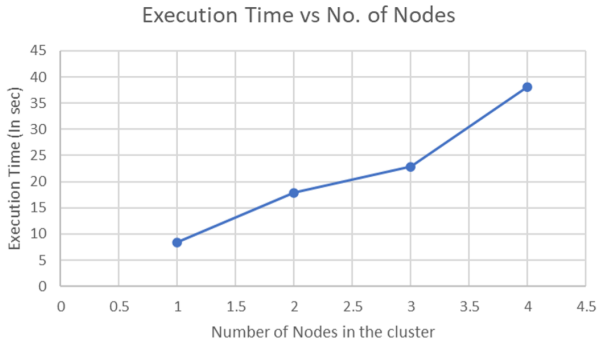


Fig. 14: Execution time vs Number of Nodes

VIII. Challenges Faced

During the implementation of federated learning on FPGA, we encountered various challenges that had to be addressed. The first challenge we faced was setting up the TCP IP communication between the FPGA boards and the server. This was necessary for transmitting the data and results between the boards and the server. Once this was established, we faced another crucial challenge - the synchronization between multiple FPGA boards and the server. It was imperative to ensure that the FPGA boards were synchronized with the server to avoid any data loss or discrepancies during the training process.

We also faced issues with the 'hls math' library functions, specifically with the 'log' function, which is essential for computing the loss during backpropagation. The issues with the 'hls math' library functions had to be resolved to ensure accurate computations.

Another challenge was precision errors that occurred during the computations on the FPGA boards. These errors had to be minimized to avoid any inaccuracies in the training results. Overall, these challenges had to be addressed to ensure that the implementation of federated learning on FPGA was successful.

IX. Future Scope

There are several opportunities to improve the performance of our federated learning system. One possible approach is to run more epochs in order to improve the accuracy of the results. Currently, the system is using 16-bit registers, but utilizing 32-bit registers could increase the precision of the results. Another potential area for improvement is training the Convolutional Layer along with the fully connected (FC) layer. By incorporating the Convolutional Layer, the system could potentially improve the accuracy of its predictions even further. These enhancements could contribute to a more robust and accurate federated learning system.

X. Timeline and Task Assignment

The following table captures the Timeline and Teammates assignment for each of the above tasks -

Task	*Assignee*	*Timeline*
Initial weights using PyTorch	Virat Agarwal Abhipsa Panigrahi	03/27/23 **Done**
Federated PS to generate average of PS Weights	Devansh Johri Khatib M. Adeeb	03/27/23 **Done**
Establishing connections between Federated PS and PL	Khatib M. Adeeb Devansh Johri	04/07/23 **Done**
Training on the PL	Abhipsa Panigrahi Virat Agarwal	04/22/23 **Done**
Multiple FPGA Communication	All 4	04/26/23 **Done**
Combining Everything	All 4	04/27/23 **Done**
Design Space Exploration	All 4	05/04/23 **Done**

TABLE I: Timeline and Responsibility Division

REFERENCES

[1] Brisimi, T. S., Chen, R., Mela, T., Olshevsky, A., Paschalidis, I. C., and Shi, W. Federated learning of predictive models from federated electronic health records. International journal of medical informatics, 112:59–67, 2018.

[2] Samarakoon, S., Bennis, M., Saad, W., and Debbah, M. Federated learning for ultra-reliable low-latency v2v communications. arXiv preprint arXiv:1805.09253, 2018.

[3] Hard, A., Rao, K., Mathews, R., Ramaswamy, S., Beaufays, F., Augenstein, S., Eichner, H., Kiddon, C., & Ramage, D. (2019, February 28). Federated Learning for Mobile keyboard prediction. arXiv.org. Retrieved March 16, 2023, from https://arxiv.org/abs/1811.03604

[4] Z. Wang et al., "PipeFL: Hardware/Software co-Design of an FPGA Accelerator for Federated Learning," in IEEE Access, vol. 10, pp. 98649-98661, 2022, doi: 10.1109/ACCESS.2022.3206785.

[5] Yang, Zhaoxiong, Shuihai Hu, and Kai Chen. "FPGA-based hardware accelerator of homomorphic encryption for efficient federated learning." arXiv preprint arXiv:2007.10560 (2020).

[6] Abreha, Haftay Gebreslasie, Mohammad Hayajneh, and Mohamed Adel Serhani. "Federated learning in edge computing: a systematic survey." Sensors 22, no. 2 (2022): 450.