Robustness of Federated Averaging Algorithms to Attacks

Iman Tabrizian
Electrical and Computer Engineering
University of Toronto
Toronto, Canada
iman.tabrizian@mail.utoronto.ca

***Abstract*—In this project, we measure the robustness of different federated averaging algorithms in response to attacks. Federated learning is vulnerable to attacks that try to make the global model converge to a different model. These attacks happen in different forms. Federated learning has given the attackers a new frontier by giving direct access to the intermediate models in the training phase. Previously, attackers only could use data poisoning techniques but now they can employ model poisoning which has more serious implications. Currently, no defense mechanism has been proposed to tackle this problem that take into account all the assumptions required in a federated learning environment. These assumptions include: non-i.i.d. environment and secure aggregation protocol. The aim of this project is to measure the level of vulnerability that is currently present in the state-of-the-art averaging algorithms.**

I. Introduction

Federated learning is a new machine learning paradigm focused on solving the privacy issues related to the machine learning. In federated learning (FL) the data is not moved from the users' device to the cloud. Instead, the model is trained on users' available data which is usually small and is non-i.i.d. This enables learning at an unprecedented scale and efficiency. Consequently, the training job is moved to the end users devices and the burden of computing is lifted from the cloud. Moreover, users' privacy has been enhanced and the communication cost is also reduced. However this method alone cannot guarantee the privacy of the users. In the real-world deployments of federated learning, this method is usually accompanied with differential privacy [1] and secure aggregation [2] mechanisms in order to prevent malicious server from accessing the individual users weights.

General model substitution in each round of federated learning can be describe using Equ. 1. In this equation, G^t represents the global model at time t and L_i^t represents the local model i at time t. n represents total number of participants and η is a hyperparameter indicating the degree to which you would like to replace the global with local one. If η equals to $\frac{n}{m}$, the global model is completely replaced with the average of local models.

$$G^{t+1} = G^t + \frac{\eta}{n}\sum_{i=1}^{m}(L_i^{t+1} - G^t) \tag{1}$$

However, there are also some problems associated with FL. Since the server cannot examine the users' data/weights, malicious users can try to decrease the accuracy or make the model mislabel some category of inputs. Even if the server could examine the weights, there are some mechanisms [3] that could be used to prevent the central server from identifying the attacks.

Algorithm 1: Global Averaging

P^1 Randomly initiate the parameters in the server;
for *round* $t \in [T]$ **do**
  The server sends P^t to all the users;
  for *worker* $w \in [n]$ **do**
    Set P^t as the initial parameters for the model;
    Train using the user's data;
    Manipulating P^t to optimize for the attacker's goals;
    Return P_i^{t+1}
  end
  ▷ Defense mechanism to prevent the attack;
  $P^{t+1} = AggregationRule(P_i^{t+1} : i \in [n])$
end

The general setup for attacking Federated learning is expressed in Fig. 1. In this setup, there are a number of devices participating in a round of federated learning. In each round, a subset of devices is selected and others only receive the weights from the central server. Since in each around a large number of users' may participate in this process, it is hard to verify that everyone is sending their correct weights. Also, because of the privacy concerns, the central server cannot examine the users' weights as it may reveal sensitive users' information. This problem has been acknowledged by the creators of federated learning [4].

$$X = G^t + \frac{\eta}{n}\sum_{i=1}^{m}(L_i^{t+1} - G^t) \tag{2}$$

In this paper, we will explore the effect of using different federated averaging algorithms on an attack on federated learning. In section II, we will explore the different types of attacks in this area, in section III we present the platform that we created during this project to be able to run these experiments. This platform is called Distributed Tensor Processing

Platform(DTP). In section **??**, we describe the challenges that we faced as a part of this project. Finally, in section **??**, we conclude the paper and mention the possible directions that this work can be extended.

II. Related Work

Attack and defense mechanisms are very important in this area of work. The place where these mechanisms are employed is described in Algorithm 1. In each round of federated learning, the server sends a list of parameters to the workers. The malicious worker tries to change P^t such that it benefits their own goal. On the other hand, defense mechanisms are employed before the aggregating all the parameters together to form the global model. The assumption in federated learning is that the federating learning server cannot inspect the weights, however, in the literature this has always been dismissed.

A. Attacking to Reduce the Accuracy

One of the most common goals of attacking a distributed learning environment is to reduce the accuracy of the model or prevent the model from converging. These types of attacks can be usually determined in the distributed deep learning environments because of the sharp drop in the accuracy and the server may decide to not include the malicious users' if they affect the performance of training.

However, in the federated learning case even a user with low accuracy may still contribute to the general model. This is the result of the non-i.i.d. data set [5].

B. Attacking to Introduce Backdoors in Federated Learning

Another type of attack is performed with the goal of introducing backdoors in the global model. This attack can be further classified into other sub categories:

1) Backdoors Involving a Pattern: In this kind of backdoors, the user tries to change the output class of images involving a certain pattern. For example, if a 5×5 black square exists on top of a MNIST digit it will be always classified as 4 regardless of the actual class. One practical benefit of this attack is to evade spam detection by including that specific pattern in the spam email.

2) Semantic Backdoors: This type of backdoors focus on misclassifying a subset of data set containing a special pattern. For example, if the car images are green they will be classified as tree.

3) Mislabeling Inputs: The idea here to misclassify a complete class of inputs. For example, all the trees will be misclassified as birds.

In these attacks, the malicious participant tries to modify the model such that it does not affect the loss function significantly and, when aggregated, the effect is not diminished. If the effect is diminished, the attacker may not be selected in the next couple of rounds and the backdoor will be averaged out in the next iterations of federated learning.

One way to attack a federated learning environment is mentioned in [5]. The goal is to introduce backdoors in the model and replace the original model with the attacker's model as described in Equ. 2. As the iterations continue the local models approach the global model but the training still keeps going. The reason is that federated learning is a continuous learning process and it never stops. In this phase of the training $\sum_{i=1}^{m-1}(L_i^{t+1} - G^t) \approx 0$. Equ. 3 suggests that the attackers local model should be scaled out by a factor of $\gamma = n/\eta$ in order to survive the global averaging. This is a good approach given that the weights changes will go unnoticed by the server.

Another attack mechanism was described in [3]. This attack is used for distributed learning environments where the federated learning server can inspect the weights. The goal here is to change the weights such that the backdoors introduced goes unnoticed. Most of the defense mechanisms described in the subsequent subsections involve working on the mean of the parameters and detecting anomaly in the mean which may correspond to malicious users. Since weights follow a normal distribution, if the attackers operate within $\pm\sigma$ of the mean, the attach will not be detected. The interesting result here is that the defense mechanisms worsen the situation and decrease the accuracy more significantly. The best defense mechanism in this case is "No Defense". Meaning that the simple averaging is the best method. However, this method cannot be always used because the attackers may use hybrid attack schemes in order to make their attack strategy effective.

By using only 25% of the users as attackers it was able to cause around 30% decrease in the model accuracy in CIFAR-10. This number of users is a large number in a federated learning context. Additionally, the assumption here is that the user will be participating in each round of averaging. These assumptions make this attack in appropriate for a federated-learning context.

The problem with almost all of the mentioned methods is that the evaluation environment barely resembles real-world scenarios. Most of the experiments are conducted in a simulated environment containing a single node. In this project, we will scale the federated environment to enable training of larger data sets and examining the effect they have in those larger data sets and models.

$$L_m^{t+1} = \frac{n}{\eta}X - (\frac{n}{\eta} - 1)G^t \approx \frac{n}{\eta}(X - G^t) + G^t \qquad (3)$$

An important note here is that the adversarial examples differ from the backdoors discussed here. Adversarial examples are the inputs that have been modified to fool the model, however, backdoors are normal inputs that will be misclassified. Backdoors are more dangerous, for example a self-driving car is trying to detect the type of the signs in a road and using an adversarial model may have serious implications on the performance of the self-driving car. The original idea that we would like to explore was to train the attacker on the adversarial data set and measure the accuracy. However, as we did more research it was mentioned that training on the adversarial examples can make the aggregated model robust to those type of examples [6]. We decided to not continue with this line of research and choose a different path.

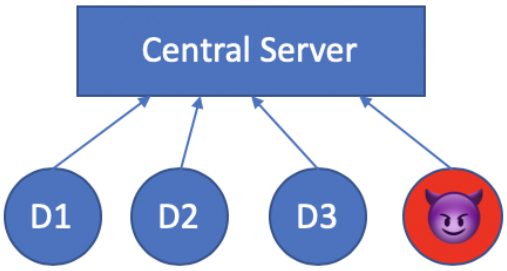

Fig. 1. Attacking Federated Learning

Now that we have discussed the methods used for attacking federated learning environment, we now aim to discuss the methods used for defense in related areas. Unfortunately, defense mechanisms generally do not apply to the federated learning context. Most of the defense mechanisms we discuss here, assume that the data distribution is i.i.dand the weights can be inspected by the central server. However, we now that because doing so will compromise the users' privacy none of the mentioned techniques neither are allowed nor can be applied because of employing Secure Aggregation [7] in most of these environments.

C. Trimmed Mean

Trimmed Mean employs the idea that malicious users' weights are far from the mean and by filtering them out we can have a safe aggregation phase. In Trimmed Mean, only the k values which are close to the mean will be selected. This mechanism cannot be employed in a federated learning context because we don't have access to the users' weights. Additionally, because federated learning is employed in a non-i.i.dcontext the weights which are far from the mean may still be applicable and do not correspond to the malicious users.

D. Krum Method

Krum is another defense mechanism used to filter out the weights reported by the malicious users. In this method, the server tries to identify a single honest participant. This participant is the user with the weights closest to $n - m - 2$ other users, where n is the total number of participants and m is the number of malicious workers. In order to defeat this defense mechanism it is enough to make the server choose the malicious participant as the honest participant. This has been achieved in [3].

E. FoolsGold

FoolsGold [8] method has been designed to mitigate the sybil attacks that is present in the federated learning environment. The argument in this method is that clients which are malicious try to change the direction of the global model to another model. This will result in the similarity in the directions proposed by the malicious clients. Additionally, this method assumes that there exist a number of honest clients which can be used in order to find the correct direction. Another assumption which makes this defense mechanism completely impractical is that secure aggregation [7] protocol is not employed so that the federated learning server can inspect the clients weights. We know that by not employing secure aggregation the data points can be guessed and the whole purpose of the federated learning is defeated.

III. Emulation Platform

During this project we developed DTP (Distributed Tensor Processing) platform based on PyTorch to enable evaluation of larger models in a federated learning environment. Currently, the platforms for Federated Learning focus on other aspects of Federated Learning. For example, Tensorflow Federated [9] focuses on federated computation in general and does not limit itself to only Federated computation. While this is a very important and prominent feature, currently, it only supports computations running on a single machine.

Another work in this area is LEAF [10]. LEAF focuses more on benchmarking federated algorithms. This is an open-source work. To achieve its goal, it introduces a set of metrics to better evaluate the federated settings. This work also runs on a single machine which can significantly limit the models and data sets that can be evaluated. Also, it has a set of data sets that present the non-i.i.d. attribute of the data sets. They have images for various tasks including image classification and next word prediction, sentiment analysis, language modeling, and next character prediction.

To address the problem of larger scale evaluations we designed a new platform based on PyTorch to experiment with the federated learning algorithms. It is simple and is implemented using around 500 lines of code. It also comes with an agent written in Golang which enables easier orchestration of the jobs across multiple machines.

The general architecture of the platform is shown in Fig. 2. Each of the agents run a DTP-Agent. DTP-Agent is an HTTP API that is used to start and kill the running tasks on each of the agents. The nodes continuously report the validation and loss using tensorboard. Tensorboard is a tool created by the authors of Tensorflow used for monitoring the ML algorithms. Also, the Gloo backend is developed by Facebook which provides constructs similar to the MPI. For example, you can send a tensor from one node to another. Also, it provides some aggregated operations such as *allreduce*. There are other communication backends apart from Gloo which can support GPU (e.g. NCCL) however they do not provide the capability to send one tensor from one place to another. In order to decrease the time required for the communication of transferring files between different nodes, we have setup a NFS between all the nodes to reduce the code upload times.

The platform contains two main stages for training of the federated model. The first step is selecting the clients that are going to participate in the training and the second case is performing the training on the locally available data. The former is explained in section III-A and the latter is explained in section III-B.

TABLE I
FEDERATED LEARNING PLATFORMS

	Federated Computation	Running on Multiple Nodes	Hardware Support
Tensorflow Federated	Yes	No	CPU/GPU/TPU
LEAF	No	No	CPU/GPU
DTP	No	No	CPU

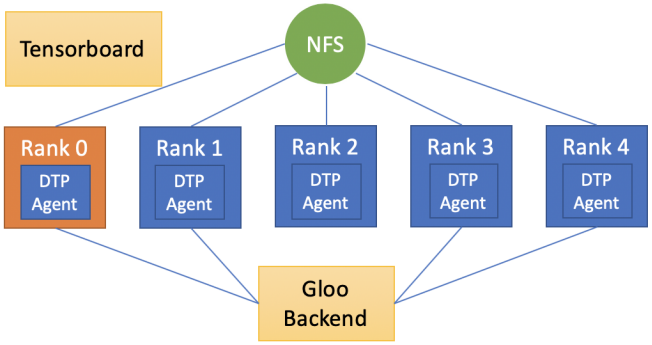

Fig. 2. Architecture of the Platform

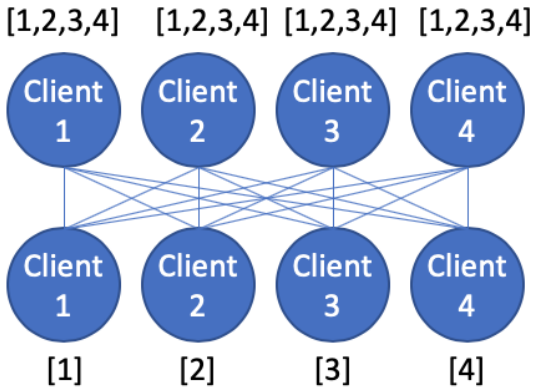

Fig. 3. Allreduce Operation in PyTorch

A. Selection Mechanism

For the selection phase, there is a single machine responsible for selecting the participating clients (selector). Depending on the type of the algorithm, this election is held in different rounds. For FedSGD, this selection should be held in every iteration, while for the FedAVG, this selection is held every i^{th} round.

The process starts by different nodes stating their rank. Rank 0 is responsible for managing the other ranks and holding selection phases. Rank 0 randomly selects e clients out of the all available n clients.

B. Averaging

The e clients will perform the averaging process for k iterations and then they will broadcast their weights to all other processes. Because in this implementation we have not used a parameter server [11], we have implemented the whole process using all reduce operations. We transfer the sum of every parameter to every other node in the cluster using allreduce and all of them will average the parameters independently. While this approach is easier to implement compared to the parameter server, it is not communication efficient. Also, it requires synchronization. There is an alternative version of allreduce named ring allreduce [12]. This approach has significantly lower cost compared to the allreduce method.

IV. RESULTS

We evaluated the robustness of the federated averaging algorithms by running 4 experiments per each data set. The first data set that we use is CIFAR-10 [13]. The second data set that we use is MNIST. MNIST is a much easier task compared to CIFAR-10. This allowed us to run more extensive experiments in the MNIST case compared to the CIFAR-10 data set.

The four common experiments that we run for each of the data sets consists of the following. First, we run the non poisoned version of each of these data sets and average the weights on each of the clients using the FedAVG and FedSGD. This serves as our baseline model to compare the effect of poisoning in federated learning. The next two experiments run the poisoned version of the data set against the FedAVG and FedSGD. The poisoning mechanism that we are currently focused on is mislabeling the targets. The attacker changes one of the targets in an input data set to another one in our example. The goal here is to explore the effectiveness of the attacks on the accuracy of the global model.

We run the platform described in section III on 5 virtual machines on the SAVI [14] testbed. Each of these VMs have 16 GBs of RAM and 8 cores. Fig. 4 shows the accuracy of the 4 experiments conducted in the previous paragraph. The model used here is ResNet-18 [15] which contains around 2 million parameters. In these experiments, one of the VMs is responsible for selecting the nodes participating in a round of federated learning. Currently only 2 out of nodes are selected in each round. Only one of the nodes have the poisoned data set which it uses to train the local model. Contrary to what we expected, the accuracy is not significantly changed. The difference between the accuracies indicate that the federated learning is intrinsically robust to the attacks. A possible reason for this is that the user with the poisoned data set participates in relatively few number of rounds. Since only 1 out 4 users have a poisoned data set, it is not very likely that this user will be selected multiple times.

Fig. 5 shows the loss of a single node having the same data

set performing under different settings. The main point here is that it seems like the loss function shows similar behavior under different settings and is not very different.

We run another set of experiments using MNIST [16]. The servers setup is similar to the previous experiment. However, in this case we explore the effect of changing the number of from attackers from one to two nodes. Fig. 6 shows the accuracy across different settings when the number of attackers is equal to two. This lead to 9.79% decrease in the accuracy in the FedSGD case and 9.57% decrease in the case of FedAVG. Also, the model architecture is different from the previous example. In this example, we designed a simple model that can achieve high accuracy on the MNIST data set. This model contains two convolutional layers and two fully connected layers. Also, it has two dropout layers to avoid over fitting. Finally, in the last layer it uses a softmax function to decide the final output.

We also changed the number of attackers from 2 to 1 to compare it with the CIFAR-10 data set. We noticed that it shows very similar characteristics compared with the 1 attacker case. The difference between the accuracies is quite similar. Also, compared to the Fig. 4 the different between the gaps are much larger which indicates that the robustness to attacks is not only related to the averaging algorithm but also related to the data set being used and the neural architecture.

To replay the attack of the algorithm mentioned in [5] we multiply the gradients of the attacker by 100 to see how this effects the algorithm. An important note here is that the poisoned data set only mislabels one of the inputs and not the other targets. This way of poisoning suggests that the maximum accuracy degradation that the attacker can achieve will be around 10%. An interesting observation here is that the accuracy is not degraded more than 10% though we have chosen 3 out of 4 participants to report false weights multiplied by 100. Fig. 8 shows the result of this experiment. It is interesting that even though the weights of the attackers are multiplied by one hundred this does not change the accuracy of the model and reduce it below 10%.

In figure 10 the time taken to run various algorithms is shown. As expected the FedAVG takes shorter time compared to the FedSGD. The reason for this is that FedSGD has to sync all the weights in every mini batch. This requires a lot of synchronization which heavily decreases the training speed and increases the training time. Also, when comparing the charts of the MNIST and CIFAR-10, it can be noticed that MNIST takes significantly less time compared to CIFAR-10. The reason is that for CIFAR-10 we have used ResNet-18 which is a significantly more complex neural net architecture compared to the SimpleNet that we created for the MNIST. More complex architecture increases the backward and forward propagation time which leads to less number of mini batches being processed every second.

When comparing the Fig. 5 with Fig. 9 we noticed that the loss function has significantly lower values when compared with the Fig. 5. The reason for this is that the MNIST is an easier task to learn and the neural net manages to find the set

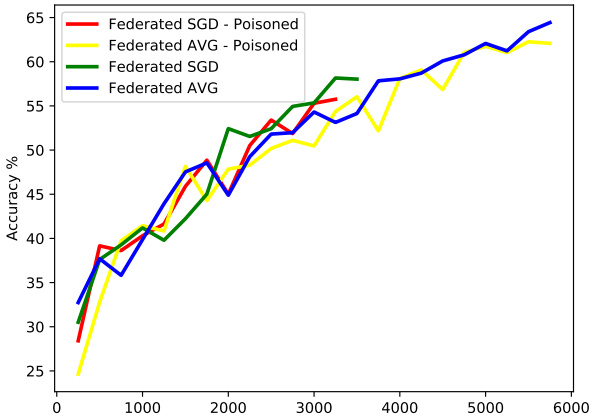

Fig. 4. Robustness of FedSGD and FedAVG to input poisoning in CIFAR-10

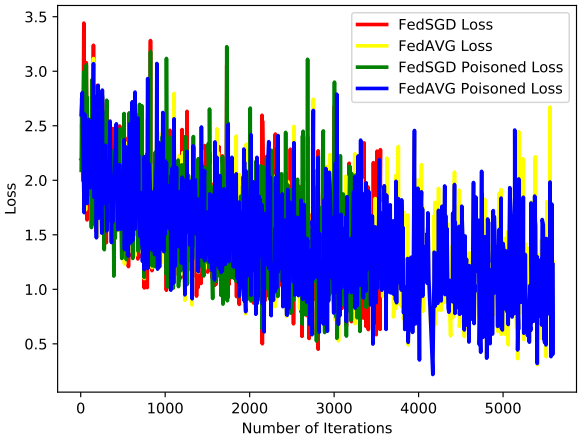

Fig. 5. Loss of the Node with Rank 2 Across Different Experiments on CIFAR-10

of the parameters that can achieve lower loss easier. This is the reason why most of the weights in the Fig. 9 are close to zero.

V. Challenges

One of the main challenges faced in this project is executing allreduce on a subset of the clients. PyTorch provides a construct named groups that allow you to group a number of nodes together. Grouping allows you to perform computation on a subset of nodes instead of all of them. The assumption when designing groups in PyTorch was that the groups are determined statically. This is a good design for the case of distributed machine learning and not federated learning. For the federated learning it is required to select a subset of nodes in each round and perform allreduce this subgroup. This requires calling the *new_group* function around 10000

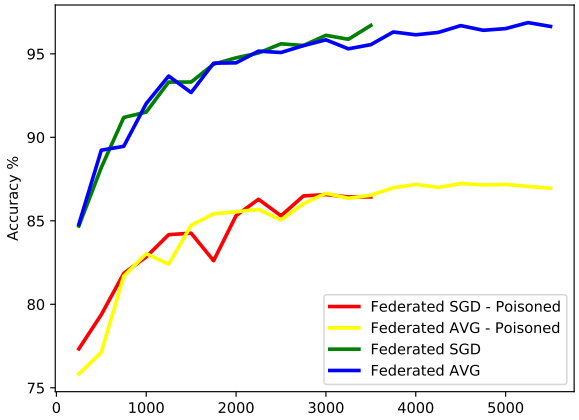

Fig. 6. Robustness of FedSGD and FedAVG to Input Poisoning in MNIST Using 2 Attackers

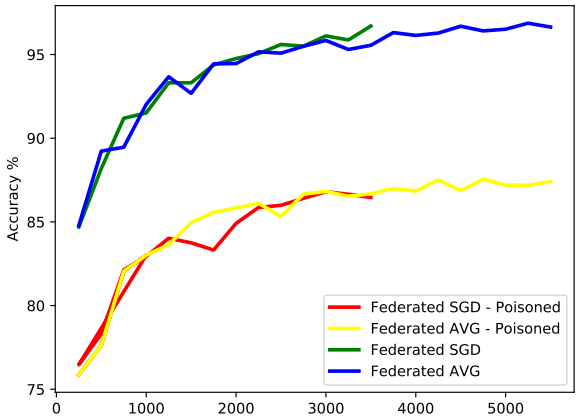

Fig. 7. Robustness of FedSGD and FedAVG to Input Poisoning in MNIST Using 1 Attacker

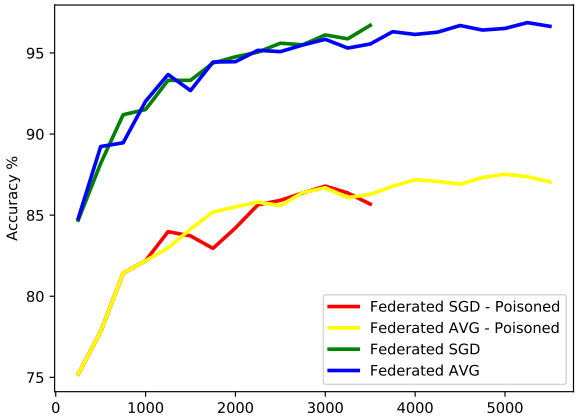

Fig. 8. Robustness of FedSGD and FedAVG to Input Poisoning When Weights are multiplied by 100 and using 3 Attackers

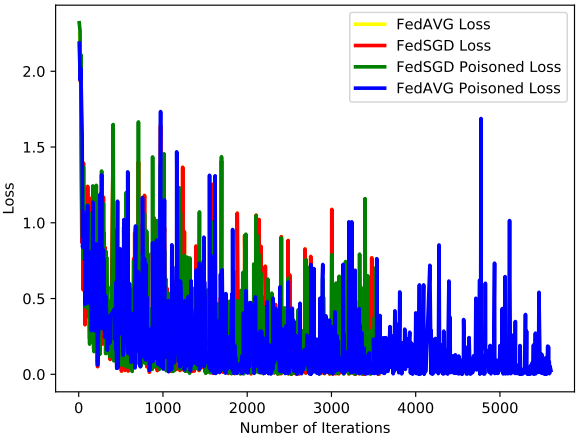

Fig. 9. Loss of the Node with Rank 2 Across Different Experiments on MNIST

times. It seems like that PyTorch does not have automatic garbage collection for deleting these groups after they have been created. It also does not provide a construct for deleting and deallocation of the groups created. This results in hitting various limits in the Linux Kernel. Another limiting factor when using the *new_group* function is that it requires that all of the nodes pass through the function even if they are not participating in that group. This also makes things a bit complicated and required reorganizing of the code in a way that facilitated this requirement.

The first problem is exceeding the limit of the number of open files. This could be easily fixed by increasing the limit to allow more file descriptors. After fixing this problem, the iteration continued for longer periods however, it again started hitting another limit. The new one is the cgroup limit. Currently, the reason for hitting this limit is not clear. cgroups are used to limit the resources that a process can consume. It is probably set by PyTorch and because of not deleting the old groups the current number of cgroups per process exceeds the limit that is available to it.

Other challenge was killing the tasks. The tasks could not be killed easily when an error or bug happens. Writing an script (which looks very simple) took a significant amount of time to filter out from all the processes and kill the ones that correspond to our platform.

VI. Conclusion and Future Work

This paper evaluated the effect of data input poisoning on the common federated averaging algorithms. A byproduct of this research is DTP. A distributed tensor processing platform that is used for fine-grained control over the training loop of

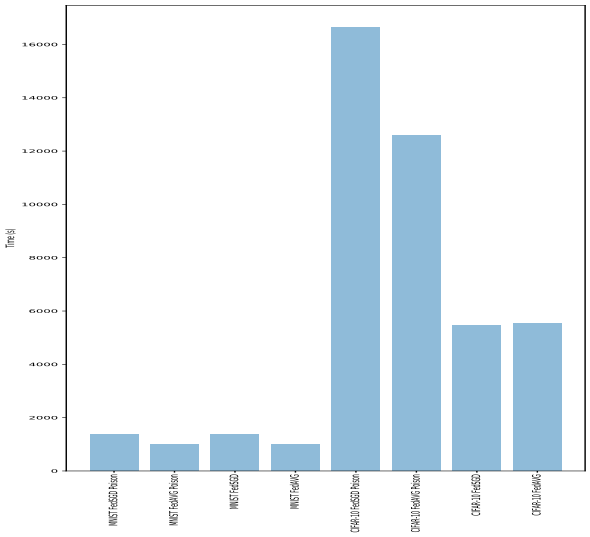

Fig. 10. Time Taken to Run the Federated Learning Algorithms

federated learning. This project was one of the first attempts to shorten the gap between the simulations and real-world deployment of federated learning.

We noticed that commonly used Federated Averaging algorithms are robust to certain extent when tackling attacks that have initiated from the clients. [5] also acknowledged that designing attacks in a federated learning context is a hard job and requires constant participation of the attacker in the training loop. This usually do not happen in real world deployments of federated learning. Since real world deployments of federated learning are usually held across large number of devices [4] which decreases the probability of the attacker being selected for the next round. We evaluated this hypothesis on two ML data sets using various configurations. Another interesting observation in this area was that we noticed that the data set and the architecture of the neural network effects the accuracy and effectiveness of attacks. We observed that the attacks were more effective on the MNIST data set with the SimpleNet architecture and lead to around %10 decrease in the accuracy. Additionally, incorporating more attackers when the federated model has already been hurt enough as a result of the attack will not decrease the model accuracy more. This result makes sense because it is equivalent of a training task were everybody shares the mislabled input data set.

A. Future Work

As mentioned in section III, DTP still has some technical problems that need to be addressed for larger scale deployments involving hundreds of machines. Also, the experiments and the data sets need to be extended to include more results and different configurations. Also, a very complex and important problem in this area of research is detecting attackers. Since the central server does not have access to the users' weights, designing a defense mechanism that can detect attacks without is a tricky problem that has not been answered yet.

Another possible direction of this work can be investigating the effect of non-i.i.d. of data sets to the effectiveness of the attacks. For example, if we can come up with a measure of how non-i.d.d. the data set is and how this effects the learning curves of the federated learning. Also, if the client containing the data set with the largest level of non-i.i.d. ness has the data set poisoned, how will this effect the training process.

This work also needs to contain a lot more nodes compared to the number of nodes that it currently has. Due to computational limitations we could experiment only to this extent. Extending the size of the experiments will significantly bring new challenges to the DTP and using efficient methods for communicating the weights will matter more. We also need to incorporate more diverse hardware platforms into the framework. Because of using Gloo backend we were limited to CPU nodes only. However, by switching to NCCL and removing the *send* and *receive* constructs, we can significantly speed up the training process and make experimentation faster in this area.

REFERENCES

[1] I. Dinur and K. Nissim, "Revealing information while preserving privacy," in *Proceedings of the twenty-second ACM SIGMOD-SIGACT-SIGART symposium on Principles of database systems*. ACM, 2003, pp. 202–210.

[2] K. Bonawitz, V. Ivanov, B. Kreuter, A. Marcedone, H. B. McMahan, S. Patel, D. Ramage, A. Segal, and K. Seth, "Practical secure aggregation for privacy-preserving machine learning," in *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*. ACM, 2017, pp. 1175–1191.

[3] M. Baruch, G. Baruch, and Y. Goldberg, "A little is enough: Circumventing defenses for distributed learning," *arXiv preprint arXiv:1902.06156*, 2019.

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

[5] E. Bagdasaryan, A. Veit, Y. Hua, D. Estrin, and V. Shmatikov, "How to backdoor federated learning," *arXiv preprint arXiv:1807.00459*, 2018.

[6] C. Szegedy, W. Zaremba, I. Sutskever, J. Bruna, D. Erhan, I. Goodfellow, and R. Fergus, "Intriguing properties of neural networks," *arXiv preprint arXiv:1312.6199*, 2013.

[7] K. Bonawitz, V. Ivanov, B. Kreuter, A. Marcedone, H. B. McMahan, S. Patel, D. Ramage, A. Segal, and K. Seth, "Practical secure aggregation for federated learning on user-held data," *arXiv preprint arXiv:1611.04482*, 2016.

[8] C. Fung, C. J. Yoon, and I. Beschastnikh, "Mitigating sybils in federated learning poisoning," *arXiv preprint arXiv:1808.04866*, 2018.

[9] "Tensorflow federated," https://www.tensorflow.org/federated/, (Accessed on 12/04/2019).

[10] S. Caldas, P. Wu, T. Li, J. Konečnỳ, H. B. McMahan, V. Smith, and A. Talwalkar, "Leaf: A benchmark for federated settings," *arXiv preprint arXiv:1812.01097*, 2018.

[11] M. Li, D. G. Andersen, J. W. Park, A. J. Smola, A. Ahmed, V. Josifovski, J. Long, E. J. Shekita, and B.-Y. Su, "Scaling distributed machine learning with the parameter server," in *11th {USENIX} Symposium on Operating Systems Design and Implementation ({OSDI} 14)*, 2014, pp. 583–598.

[12] A. Gibiansky, "Bringing hpc techniques to deep learning -," http://andrew.gibiansky.com/, (Accessed on 12/04/2019).

[13] A. Krizhevsky, V. Nair, and G. Hinton, "The cifar-10 dataset," *online: http://www. cs. toronto. edu/kriz/cifar. html*, vol. 55, 2014.

[14] J.-M. Kang, H. Bannazadeh, and A. Leon-Garcia, "Savi testbed: Control and management of converged virtual ict resources," in *2013 IFIP/IEEE International Symposium on Integrated Network Management (IM 2013)*. IEEE, 2013, pp. 664–667.

[15] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2016, pp. 770–778.

[16] P. J. Grother, "Nist special database 19," *Handprinted forms and characters database, National Institute of Standards and Technology*, 1995.

APPENDIX

This section provides information about the software and hardware requirements that enabled this project. Throughout the experiments mentioned in this paper we used Ubuntu 18.04 as the base image. The list below contains the necessary software which needs to be installed on top of the Ubuntu image. You need at least 5 VMs with at least 4 GBs of RAM and 4 cores in order to run the experiments efficiently.

After setting up Miniconda on each of the VMs run the commands specified in the listing 1

Listing 1. Install Python Packages

```
1 pip install tensorboard==2.1.0
2 conda install pytorch torchvision -c pytorch
```

Software requirements of the machine that is used to SSH into the remote machines is listed in Table III.

Now you need to setup the NFS server. Choose one of the VMs as the NFS server. Run the script *setup-nfs.sh* from the https://github.com/tabrizian/dtp on the selected VM. Make sure to replace the line subnet in line 2 with the appropriate subnet that you want to able to access the NFS server (usually the subnet which the VMs are using).

Now run the *setup-nfs-client.sh* on the remaining VMs. Make sure to change the last line with the IP address of the NFS server. Make sure the firewall allows the servers to transfer data on any TCP and UDP port.

The main file that we will be using for the experiments is main.py. Different configurations can be easily applied by changing the values in the beginning of the file. For example, you can change the value of *LOCAL_ITERATION* from 1 to 5, you switch from FedSGD to FedAVG. Also, if you do not want to introduce any attacks in your model you can set the value of *fake_targets* variable to an empty array. Also, make sure to change the *writer = SummaryWriter(path)* in every run of the application so that you do not overwrite the old values.

To run the code type *python3 main.py rank total_vms*. Replace the *rank* with the number that you have assigned as the rank of the machine and *total_vms* as the total number of virtual machines participating in the experiment. Now you need to wait for the algorithm to run to completion. After the process finishes the logs are available in the path you specified as the argument of *SummaryWriter* you can view the logs using this command: *tensorboard –logdir path –port 6006 –bind_all*. Now go to the browser on the *http://ip:6006*. You should be able to view the accuracy and loss on each of the nodes participating in the experiment. Note that the current implementation only calculates the accuracy on the node with rank 1.

TABLE II
SOFTWARE REQUIREMENTS OF THE VMS

Software Name	Version	Link
Miniconda	4.7.12	https://docs.conda.io/en/latest/miniconda.html#linux-installers
Tensorboard	2.1.0	https://pypi.org/project/tensorboard/
PyTorch	1.3	https://pytorch.org/get-started/locally/

TABLE III
SOFTWARE REQUIREMENTS OF THE VMS

Software Name	Version	Link
pssh	2.3.1	https://pypi.org/project/pssh/