RESPIPE: RESILIENT MODEL-DISTRIBUTED DNN TRAINING AT EDGE NETWORKS

Pengzhen Li, Erdem Koyuncu, and Hulya Seferoglu
pli33@uic.edu, ekoyuncu@uic.edu, hulya@uic.edu

Department of Electrical and Computer Engineering
University of Illinois at Chicago

ABSTRACT

The traditional approach to distributed deep neural network (DNN) training is data-distributed learning, which partitions and distributes data to workers. This approach, although has good convergence properties, has high communication cost, which puts a strain especially on edge systems and increases delay. An emerging approach is model-distributed learning, where a training model is distributed across workers. Model-distributed learning is a promising approach to reduce communication and storage costs, which is crucial for edge systems. In this paper, we design ResPipe, a novel resilient model-distributed DNN training mechanism against delayed/failed workers. We analyze the communication cost of ResPipe and demonstrate the trade-off between resiliency and communication cost. We implement ResPipe in a real testbed consisting of Android-based smartphones, and show that it improves the convergence rate and accuracy of training for convolutional neural networks (CNNs).

***Index Terms*—** Deep neural networks (DNN), distributed training, edge networks, resiliency.

1. INTRODUCTION

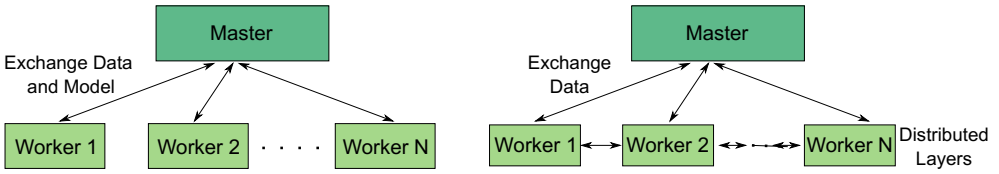
The traditional approach to distributed deep neural network (DNN) training is *data-distributed* learning, which partitions and distributes data to workers as illustrated in a master/worker setup in Fig. 1(a). In this setup, the data is partitioned as many times as there are workers in the system and all workers train multiple instances of the same model on different subsets of the training dataset. The workers apply the same algorithm to different datasets. The same model is available to all workers after updated at the master device. Data-distributed learning (also called data parallelism) has been frequently used as a de-facto distributed learning mechanism in practical systems, especially in multi-GPU platforms. This approach, although has good convergence properties, has high communication and storage costs, which puts a strain especially on edge systems. Excessive communication among computing entities due to frequent weight synchronization in data-distributed learning increases transmission delay and overall training delay. Furthermore, computing entities should receive and store all the weights, which may not be feasible for large models when storage is limited.

Model-distributed learning is an emerging approach, where parts/layers of a training model are distributed across workers, Fig. 1(b). In this setup, no single worker has a complete model, but it is distributed across all workers. Thanks to distributing model across multiple devices, the whole model does not need to be exchanged among master and workers, which is promising to reduce communication and storage costs. This approach has been recently investigated in multi-GPU platforms, [1–3].

Distributed DNN training in edge systems has unique challenges as compared to multi-GPU platforms due to the heterogeneous and dynamic nature of edge computing systems and resources as well as unreliable and delayed computing entities. Furthermore, model-distributed learning is more vulnerable to delayed and failing workers as compared to data-distributed learning, because workers depend on each other to collectively train the distributed model as seen in Fig. 1.

In this paper, we focus on model-distributed DNN training at edge networks where edge devices may be delayed or fail during training. We design *ResPipe*, a novel resilient model-distributed DNN training algorithm, where layers of a DNN training model instead of

This work was supported in parts by the Army Research Lab (ARL) under Grant W911NF-1820181, National Science Foundation (NSF) under Grants CCF-1942878 and CCF-1814717, the National Institute of Standards and Technology (NIST) under Grant 70NANB17H188, and the University of Illinois at Chicago Discovery Partners Institute Seed Funding Program.



(a) Data-Distributed Learning (b) Model-Distributed Learning

Fig. 1: Data- and Model-Distributed Learning

data are distributed across workers. ResPipe is resilient against delayed/failing workers thanks to proactively exchanging weights of the model among workers. We analyze the communication cost of ResPipe, and show that there is a trade-off between communication cost and resiliency. We implement ResPipe in a testbed consisting of Android-based smartphones, and show that it improves the convergence rate and accuracy as compared to baselines; data-distributed training and PipeDream [1] for convolutional neural networks (CNNs).

2. RELATED WORK

Model distributed learning has been recently investigated in multi-GPU platforms to overcome communication and storage limitations and provide scalability for DNN. GPipe [2] uses pipeline parallelism for communication efficient distributed DNN training. PipeDream [1] improves the efficiency by eliminating the idle workers. A weight prediction technique is developed in [3] to improve accuracy of [1, 4]. As compared to this line of work, ResPipe focuses on resilient model-distribution.

Communication cost reduction techniques for distributed learning such as gradient sparsification [5, 6], truncation [7, 8], quantization [9, 10] have been widely studied in the literature. Our work is complementary to this line of work, where communication among workers can still be reduced using gradient sparsification, truncation, or quantization. Existing work at the intersection of edge computing and machine learning usually focuses on resource allocation and optimization across edge devices to accelerate convergence speed [11–14]. As compared to this line of work, our focus is on model-distributed learning in edge computing systems, and our work is complementary to resource allocation mechanisms. Another related area of work is federated learning, which aims to train models across multiple decentralized edge devices, without exchanging data samples [15–18]. Our system setup, where data is collected by the master device, is different than federated learning, where data is naturally distributed across edge devices.

3. MODEL

Setup. We consider a distributed computing system formed of connected computing entities; end devices, edge servers, and cloud. We divide these computing entities into *(i) masters* who want to perform intensive computations on their collected data; or *(ii) workers* who are willing to dedicate some of their resources to help in the computations. There could be multiple masters and workers in the system, which may overlap. We consider a multi-hop network where each device in the network is responsible with relaying and computing offloaded tasks.

Master/Worker Model. We focus on a master/worker setup at the edge network, where the master device offloads its computationally intensive tasks to Worker $n \in \mathcal{N}$ (where $\mathcal{N} \triangleq \{1, \dots, N\}$) via device-to-device (D2D) links such as Wi-Fi Direct and/or Bluetooth. Computationally intensive tasks are determined by DNNs.

The workers have the following properties: *(i) Failures:* Workers may fail or "sleep/die" or leave the network before finishing their assigned computational tasks. *(ii) Stragglers:* Workers incur probabilistic delays in responding to the master, where delay has two components; transmission delay for exchanging data, model, and parameters, and computation delay.

Learning Model. We denote the training dataset by $A^T \in \mathbb{R}^{m \times d}$, the label vector by $\mathbf{z} \in \{0, 1\}^m$, and the weight vector of the model by $\mathbf{x}$. The weight vector of a DNN is obtained by minimizing the loss function $C(\mathbf{x})$, which is determined by the learning model, via gradient descent [19]; $\mathbf{x}^{j+1} = \mathbf{x}^j - \eta \nabla(C(\mathbf{x}^j))$, where $\mathbf{x}^j$ is the weight vector at the j^{th} iteration, η is the learning rate, $\nabla(C(\mathbf{x}^j))$ is the gradient of $C(\mathbf{x}^j)$.

4. RESILIENT MODEL-DISTRIBUTION

4.1. Model-Distributed Learning via Pipelining

In model-distributed learning, model $\mathbf{x}$ is partitioned and distributed across workers. Each worker stores and updates a subset of the model. Workers process data A, calculates activation vector in the forward propagation phase and error vector for back-propagation phase. These vectors are exchanged among workers, which reduces communication cost as compared to data-distributed learning. The main idea of model-distributed learning is provided next via an illustrative example.

Example 1 *Let us consider that we train a* 4*-layer fully connected neural network with dataset* $A_{(1)}, \dots A_{(m)}$.

There are K neurons in each layer. The goal is to learn the model $\mathbf{x} = \{\mathbf{x}_l\}_{l=1}^{4}$, where $\mathbf{x}_l$ is a $K \times K$ matrix. There are 4 workers in this setup, and each layer is assigned to one worker, i.e., layer l is assigned to Worker l, so Worker l keeps and updates the weight matrix $\mathbf{x}_l$.

In model-distributed learning, Worker 1 receives data $A_{(j)}$ from the master device (or already has the data), calculates $\mathbf{a}_1^j = A_{(j)}$ and $\mathbf{u}_1^j = \mathbf{x}_1^j \mathbf{a}_1^j$. It transmits $\mathbf{u}_1^j$ to Worker 2. Similarly, Worker 2 and Worker 3 calculate $\mathbf{a}_n^j = g(\mathbf{u}_{n-1}^j)$ and $\mathbf{u}_n^j = \mathbf{x}_n^j \mathbf{a}_n^j$, where $n \in \{2, 3\}$ and g is an activation function. Worker 2 sends $\mathbf{u}_2^j$ to Worker 3 and Worker 3 sends $\mathbf{u}_3^j$ to Worker 4, which calculates $\mathbf{a}_4^j = g(\mathbf{u}_3^j)$. This completes the forward-propagation.

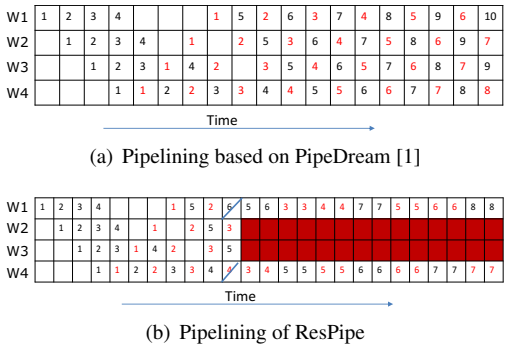
In the back-propagation phase, Worker 4 calculates $\mathbf{v}_4^j = \mathbf{a}_4^j - \mathbf{y}^j$, where $\mathbf{y}_j$ is the output vector, and sends $\mathbf{v}_4^j$ to Worker 3. Workers 2 and 3 calculate $\mathbf{v}_n^j = (\mathbf{x}_n^j)^T \mathbf{v}_{n+1}^j . * g'(\mathbf{u}_{n-1}^j)$, where $n \in \{2, 3\}$, $.*$ is an element-wise multiplication, and g' is the derivative of g. Worker 3 sends $\mathbf{v}_3^j$ to Worker 2, and Worker 2 sends $\mathbf{v}_2^j$ to Worker 1. Then, Workers 1, 2, and 3 calculate gradient vectors as $\Delta_n^j = \Delta_n^j + \mathbf{v}_{n+1}^j (\mathbf{a}_n^j)^T$, where $n \in \{1, 2, 3\}$, and $\nabla C(\mathbf{x}_l^j) = \Delta_l^j$. This completes the back-propagation phase.

Finally, each worker updates its model according to $\mathbf{x}_l^{j+1} = \mathbf{x}_l^j - \eta \nabla C(\mathbf{x}_l^j)$, $l \in \{1 \dots 4\}$. The model is updated in a distributed manner across all workers. As seen, only vectors $\mathbf{u}_l^j$ and $\mathbf{v}_l^j$ are exchanged among workers, not the model itself. □

Pipelining is crucial in model-distributed learning so that workers should not be idle waiting for the updates of other workers. For example, Worker 1 in Example 1 stays idle between transmitting $\mathbf{u}_1^j$ and receiving $\mathbf{v}_2^j$, which is not efficient. Pipelining addresses this issue by keeping workers busy all the time. An example pipelining procedure based on PipeDream [1] is demonstrated in Fig. 2(a) for Example 1. We note that the gradients are computed with delayed weights due to pipelining [1]. A crucial challenge of the model-distributed DNN training is the heterogeneous and time-varying resources of edge devices including computing power, storage, battery, bandwidth, etc. Furthermore, straggling (delayed) workers or failures potentially affect the performance of model-distributed learning. For example, if Worker 2 in Example 1 is delayed (i.e., if it is a straggler), this delays all the calculations. Thus, it is imperative to develop a resilient model-distributed DNN training framework.

4.2. ResPipe

ResPipe provides resiliency by introducing redundant weight exchanges among workers. The main idea



(a) Pipelining based on PipeDream [1]

(b) Pipelining of ResPipe

Fig. 2: Pipelining for (a) PipeDream [1] and (b) ResPipe. Wi is the ith worker in the system. Numbers in the boxes indicate the data/minibatch ID. The black color represents forward propagation, while red color represents back-propagation. We assume that forward and backward works take one time unit. Red boxes in (b) indicate unresponsive workers (2nd and 3rd workers). The crossed out boxes in blue color corresponds to lost weights due to failed workers.

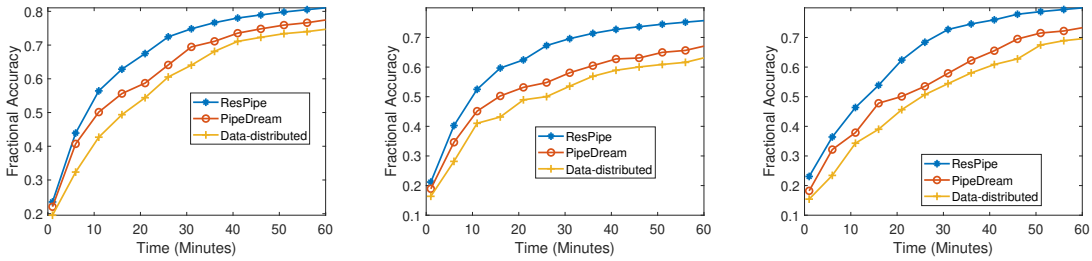
of ResPipe is provided in the next example.

Example 2 *Let us consider the same setup in Example 1. To address the issue of failing workers, more weights are exchanged among workers. Assume that only one worker fails, but we do not know which worker. In this scenario, each worker sends its weight matrix to its neighboring worker periodically (less frequently than forward propagation updates). In particular, ith worker sends $\mathbf{x}_i$ to $i+1$th worker, $i+1 \leq 4$. In this scenario, even if one worker fails, the system keeps training the DNN model. Fig. 2(b) demonstrates how ResPipe recovers from two failed workers. As seen, 1st and 4th workers keep training the model (although it takes 2 time units to finish each forward and backward task).*

Let us assume that z workers out of N could be delayed or failed during DNN training. Each worker in ResPipe keeps track of their z immediate neighbor workers[1], and send their weights to these z workers at every T time units. The next theorem characterizes the communication cost of data-distributed learning, model-distributed learning, and ResPipe.

Theorem 1 *Assume a fully connected L-layer neural network with K neurons in each layer, and there are N workers. The communication cost of data-distributed learning is $O(NLK^2)$, while it is $O(NK)$ for model-distributed learning with pipelining. On the other hand*

[1]In our Android testbed, the master device keeps track of workers, and informs each worker about their z-immediate neighbors. Peer discovery mechanisms can be applied in larger and decentralized systems.



(a) Three workers. 5 min ON, 1.5 min OFF (b) Three workers. 5 min ON, 5 min OFF (c) Five workers. 5 min ON, 5 min OFF

Fig. 3: Distributed training of a CNN for three and five worker scenarios. One worker fails in both scenarios.

the communication cost of ResPipe is $O(\frac{zLK^2}{T})$*, where* z *is the number of delayed/failed workers, and* T *is the period of updating redundant weights.*

Proof. The proof is provided in [20].

Theorem 1 demonstrates that the communication cost of ResPipe is lower than data-distributed learning. The communication cost of ResPipe depends on the number of delayed/failing workers (z) as well as the period of exchanging redundant weights (T). Noting that z and T are the parameters that determines the resiliency of the system, we can conclude that there is a tradeoff between resiliency and communication cost. Also, ResPipe introduces higher communication overhead as compared to model-distributed learning, but it provides resiliency, which is crucial for the convergence and accuracy of the training model as demonstrated next.[2]

5. EXPERIMENTAL EVALUATION

In this section, we demonstrate the performance of ResPipe in a real testbed consisting of Android-based smartphones. We consider a topology with one master and three and five workers in two different scenarios. The master device is a Google Nexus 5, and the worker devices are all Google Nexus 6Ps. Since, for this set of experiments, the master node will not need do perform intensive computations, we have utilized the relatively weaker Google Nexus 5 as the master node. Master and worker devices are connected to each other via WiFi Direct. We demonstrate the performance of ResPipe as compared to model-distibuted (specifically PipeDream [1]) and data-distributed learning.

Fig. 3(a) shows the accuracy of classification of the trained model versus the time elapsed in minutes during training for the distributed training of a CNN over the MNIST dataset. The input is a 28×28 grayscale image. Beginning with the first layer, the CNN layers are given by the sequence C_8, M_2, C_{16}, M_2, C_{32}, M_2, F_{128}, F_{10}, where C_i represents a convolution layer with i $3 \times 3 \times k$ filters applied to all k channels of the preceding layer, M_2 is a 2×2 max-pooling layer, and F_j represent a fully connected layer with j neurons. All convolution layers as well as the fully connected layers are followed by sigmoid activations. In this setup, the second worker operates at its full performance (ON mode) for 5min and fails for 1.5 min (OFF mode). This ON/OFF mode repeats itself. Initially CNN Layers C_8 and M_2 are located at Worker 1, Layers C_{16} and M_2 are located at Worker 2, and Layers C_{32}, M_2, F_{128}, and F_{10} are located at Worker 3. The redundant weight exchange period of ResPipe is $T = 30$sec. As seen, ResPipe improves over data-distributed training thanks to reducing communication cost by distributing model itself rather than data. ResPipe also improves over PipeDream thanks to providing resiliency (even though the communication cost of ResPipe is higher). The improvement of ResPipe increases when OFF duration of the failing worker increases from 1.5 min to 5 min as seen in Fig. 3(b).

Fig. 3(c) shows accuracy versus time graph for five worker scenario. The CNN model is C_{16}, M_2, C_{16}, C_{32}, M_2, C_{64}, F_{256}, F_{100}, F_{10}. The second worker fails for 5 min at every 10min. ResPipe improves both convergence time and accuracy as compared to PipeDream and data-distributed learning thanks to providing resiliency.

6. CONCLUSION

In this paper, we designed ResPipe, a resilient model-distributed DNN training mechanism against delayed/failed workers. We analyzed the communication cost of ResPipe and demonstrated the trade-off between resiliency and communication cost. The experimental results in a real testbed consisting of Android-based smartphones shows that ResPipe improves the convergence rate and accuracy significantly as compared to baselines; PipeDream and data-distributed learning.

[2]We show in the extended version [20] that the storage cost of ResPipe has similar characteristics to its communication cost as compared to data and model-distributed training

7. REFERENCES

[1] A. Harlap, D. Narayanan, A. Phanishayee, V. Seshadri, N. Devanur, G. Ganger, and P. Gibbons, "Pipedream: Fast and efficient pipeline parallel dnn training," *arXiv preprint arXiv:1806.03377*, 2018.

[2] Y. Huang, Y. Cheng, A. Bapna, O. Firat, D. Chen, M. Chen, H. Lee, J. Ngiam, Q. V. Le, Y. Wu *et al.*, "Gpipe: Efficient training of giant neural networks using pipeline parallelism," in *Advances in Neural Information Processing Systems*, 2019, pp. 103–112.

[3] C.-C. Chen, C.-L. Yang, and H.-Y. Cheng, "Efficient and robust parallel dnn training through model parallelism on multi-gpu platform," *arXiv preprint arXiv:1809.02839*, 2018.

[4] Y. Huang, Y. Cheng, D. Chen, H. Lee, J. Ngiam, Q. V. Le, and Z. Chen, "Gpipe: Efficient training of giant neural networks using pipeline parallelism," *arXiv preprint arXiv:1811.06965*, 2018.

[5] J. Wangni, J. Wang, J. Liu, and T. Zhang, "Gradient sparsification for communication-efficient distributed optimization," in *Advances in Neural Information Processing Systems*, 2018, pp. 1299–1309.

[6] D. Alistarh, T. Hoefler, M. Johansson, N. Konstantinov, S. Khirirat, and C. Renggli, "The convergence of sparsified gradient methods," in *Advances in Neural Information Processing Systems*, 2018, pp. 5973–5983.

[7] J. Langford, L. Li, and T. Zhang, "Sparse online learning via truncated gradient," *Journal of Machine Learning Research*, vol. 10, no. Mar, pp. 777–801, 2009.

[8] X. Yuan, P. Li, and T. Zhang, "Gradient hard thresholding pursuit for sparsity-constrained optimization," in *International Conference on Machine Learning*, 2014, pp. 127–135.

[9] D. Alistarh, D. Grubic, J. Li, R. Tomioka, and M. Vojnovic, "Qsgd: Communication-efficient sgd via gradient quantization and encoding," in *Advances in Neural Information Processing Systems*, 2017, pp. 1709–1720.

[10] W. Wen, C. Xu, F. Yan, C. Wu, Y. Wang, Y. Chen, and H. Li, "Terngrad: Ternary gradients to reduce communication in distributed deep learning," in *Advances in neural information processing systems*, 2017, pp. 1509–1519.

[11] S. Wang, T. Tuor, T. Salonidis, K. K. Leung, C. Makaya, T. He, and K. Chan, "When edge meets learning: Adaptive control for resource-constrained distributed machine learning," in *IEEE INFOCOM 2018-IEEE Conference on Computer Communications*. IEEE, 2018, pp. 63–71.

[12] ——, "Adaptive federated learning in resource constrained edge computing systems," *IEEE Journal on Selected Areas in Communications*, vol. 37, no. 6, pp. 1205–1221, 2019.

[13] H. Li, K. Ota, and M. Dong, "Learning iot in edge: Deep learning for the internet of things with edge computing," *IEEE network*, vol. 32, no. 1, pp. 96–101, 2018.

[14] K. Portelli and C. Anagnostopoulos, "Leveraging edge computing through collaborative machine learning," in *2017 5th International Conference on Future Internet of Things and Cloud Workshops (FiCloudW)*. IEEE, 2017, pp. 164–169.

[15] J. Konečnỳ, H. B. McMahan, F. X. Yu, P. Richtárik, A. T. Suresh, and D. Bacon, "Federated learning: Strategies for improving communication efficiency," *arXiv preprint arXiv:1610.05492*, 2016.

[16] V. Smith, C.-K. Chiang, M. Sanjabi, and A. S. Talwalkar, "Federated multi-task learning," in *Advances in Neural Information Processing Systems*, 2017, pp. 4424–4434.

[17] 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.

[18] Q. Yang, Y. Liu, T. Chen, and Y. Tong, "Federated machine learning: Concept and applications," *ACM Transactions on Intelligent Systems and Technology (TIST)*, vol. 10, no. 2, pp. 1–19, 2019.

[19] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," in *Advances in neural information processing systems*, 2012, pp. 1097–1105.

[20] P. Li, E. Koyuncu, and H. Seferoglu, "Respipe: Resilient model-distributed dnn training at edge networks." [Online]. Available: https://nrl.ece.uic.edu/index.html