Rethinking Transport Layer Design for Distributed Machine Learning

Jiacheng Xia[1] Gaoxiong Zeng[1] Junxue Zhang[1,2] Weiyan Wang[1]
Wei Bai[3] Junchen Jiang[4] Kai Chen[1,5]

[1]SING Lab, Hong Kong University of Science and Technology
[2]CLUSTAR [3]MSRA [4]University of Chicago [5]Peng Cheng Lab

ABSTRACT

Motivated by the increasing scale of data, we see a growing need of high performance distributed machine learning systems. Many research works are being proposed to improve distributed machine learning performance.

In this paper, we call upon this community to rethink transport layer solutions for distributed machine learning due to their stringent network requirements and special algorithmic properties. Distributed machine learning jobs generate bursty traffic when synchronizing parameters and a long tail flow can significantly slow down the complete training process. Meanwhile, in contrast to other distributed system applications, we find that machine learning algorithms are *bounded-loss tolerant*: randomized network data losses below a certain fraction (typically 10%–35%) will do little harm to the end to end job performance. Motivated by this observation, we highlight new opportunities to design bounded-loss tolerant transport to optimize the performance of distributed machine learning. By intentionally ignoring some packet losses, we can avoid unnecessary loss retransmissions, thus reducing the tail flow completion time. Following this principle, our preliminary results show that a simplified protocol can give 1.1–2.2x speedup on different distributed machine learning tasks.

CCS CONCEPTS

• **Networks** → *Network protocols.*

APNet '19, August 17–18, 2019, Beijing, China

ACM ISBN 978-1-4503-7635-8/19/08…$15.00
https://doi.org/10.1145/3343180.3343186

KEYWORDS

network protocols, distributed machine learning

ACM Reference Format:
Jiacheng Xia[1] Gaoxiong Zeng[1] Junxue Zhang[1,2] Weiyan Wang[1] and Wei Bai[3] Junchen Jiang[4] Kai Chen[1,5]. 2019. Rethinking Transport Layer Design for Distributed Machine Learning. In *3rd Asia-Pacific Workshop on Networking 2019 (APNet '19), August 17–18, 2019, Beijing, China.* ACM, New York, NY, USA, 7 pages. https://doi.org/10.1145/3343180.3343186

1 INTRODUCTION

Modern machine learning trains complex models (*e.g.,* deep neural networks) by iteratively applying stochastic gradient descent (SGD) to refine the model parameters [7]. This process, however, is known to be frustratingly slow: despite being easily parallelizable, the training algorithms must synchronize massive amounts of parameter updates across distributed workers (10s to 100s) at the end of each iteration, creating potential congestion and in the worst case, 1-2 orders of magnitudes of slowdowns on the entire training process.

Therefore, trimming the communication overhead for ML training has gained much attention, in both ML and systems communities. Many ML-inspired solutions exploit the property that SGD-based training produces equally accurate models, even in the presence of a non-trivial amount of delayed, missed or lower-precision updates [4, 17, 23, 24]. In the meantime, systems community has explored alternative communication patterns to mitigate traffic hotspots [16, 20, 25]. While being effective, neither approach addresses the real culprit: *tail flow completion time*, which are caused by transient traffic bursts (due to self-inflicted or competing applications) and plague modern high-speed datacenters [5, 26]. Thus, compressing parameter updates or optimizing parameter communication patterns is necessarily incomplete.

Why a transport-layer solution? In this paper, we argue that a *transport-layer* solution is fundamentally needed to trim the communication overhead in ML training. The key insight is that we can completely avoid tail flow completion time, if the transport layer simply ignores the fraction of

data delayed or missed by the network. First, doing so will not affect the training outcome (*i.e.,* same final accuracy and same number of iterations), as long as the amount of ignored data does not exceed some threshold (typically, 10%–35%). Second, default transport protocols only offer "all-or-nothing" semantics, leaving any solutions above transport layer unable to circumvent the tail latency. Note that these arguments do not apply to general data analytic applications (*e.g.,* MapReduce) in which data communication must be reliable.

Why a new transport-layer design? These desirable behaviors, however, do not match existing transport-layer techniques. The mismatch stems from the *bounded loss tolerance* property exhibited by SGD-based training—the utility of a flow is determined by the time needed to transfer a predefined fraction $p < 100\%$ of all data, not by the time to transfer all data. In contrast, TCP and its variants seek to minimize the time to transfer all data ($p = 100\%$), which inevitably suffers from tail latency even under a tiny fraction of packet delays/losses. UDP, on the other hand, does tolerate tail effect, but without the guarantee that at least p data must be delivered, which can result in poor eventual accuracy and/or require more iterations to converge.

In the rest of this short paper, we start with understanding the initial promise of a better transport-layer approach. Using large scale NS-3 simulations, we show a bounded-loss tolerance transport protocol can produce 1.1-2.2x speedup on different ML tasks under simplified assumptions. The simplified protocol significantly reduces the tail latency by tolerating bounded packet losses. With an analysis on the message contents of distributed ML jobs, we show a bounded-loss protocol can provide valid information for ML applications in presence of discrete packet drops. We then discuss related works and call upon this community to develop transport layer solutions for distributed ML leveraging such bounded-loss tolerance feature.

2 MOTIVATION

We first explain how SGD-based algorithms for ML works and point out that distributed ML suffer from tail latency (§2.1), describe the bounded loss tolerant feature of distributed ML (§2.2), and address that application level optimizations cannot fully benefit from such a feature (§2.3). This motivates us to design new transport layer protocols leveraging bounded-loss tolerance.

2.1 Overview of distributed ML

The typical goal of machine learning is minimizing an objective function value. This value measures the performance of the ML model, for example, represents the error rate for a classification problem. In minimizing the objective value,

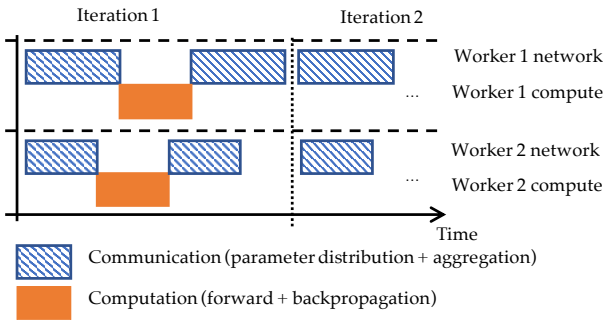


Figure 1: Common pattern of distributed ML jobs. Workers need to wait for other workers to finish the current iteration before starting the next iteration.

Model	RNN	MLP	MF	CNN
Slowdown (JCT-tail / JCT-avg)	1.1	1.77	1.93	1.32

Table 1: TCP's tail FCT slows down distributed ML

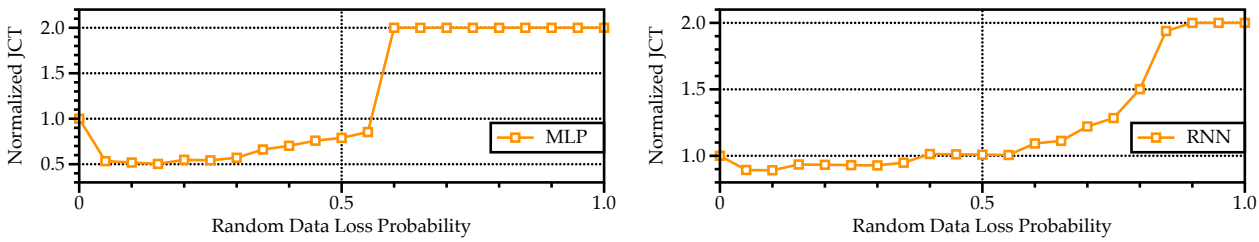
SGD-based algorithms [7, 14, 18] are used. Such algorithms run for multiple iterations, in each iteration, it measures the objective value and adjusts the model to greedily reduce objective value in the following iteration.

The key metric for measuring the quality of machine learning job is *convergence*, i.e., model loss on validation dataset (a part of data not involved in training the model) to be stably low [12]. A lower loss value at convergence represents better application level performance.

To handle huge datasets, synchronized distributed training is often used. In distributed training process, multiple worker nodes train the same model. In each iteration, workers fetch the current parameters of the models, trains the model locally and exchange their results to update a global model, as figure 1 shows. The major content in communication in distributed ML can be viewed as a vector of floating point numbers describing the model.

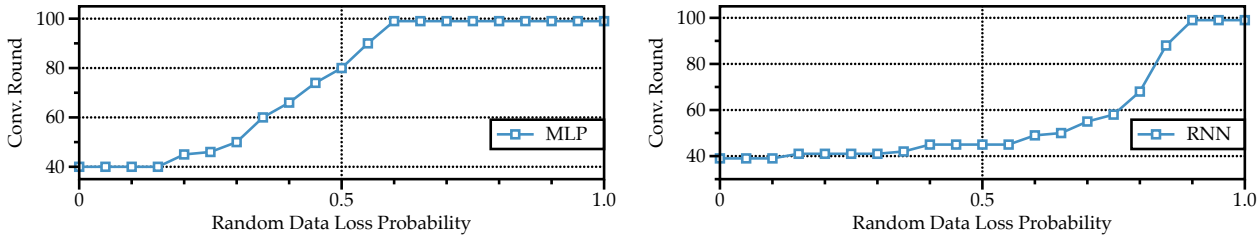
Distributed ML suffers from tail latency. To keep workers' model up-to-date, model is synced in every iteration. At the beginning and end of each iteration, multiple flows are generated almost simultaneously to exchange data among workers, generating bursty network traffic. Meanwhile, many ML models are trained under strong synchronization requirements, i.e., all workers need to update their parameters prior to starting next iteration. Therefore, the performance is determined by the tail completion time of all the flows in one iteration.

Under existing transport protocols, the tail completion time is much larger than average. As existing ML frameworks rely on RDT protocols like TCP to transmit messages [3, 20],



(a) Loss-JCT tradeoff of MLP. JCT limit= 2

(b) Loss-JCT tradeoff of RNN



(c) Loss-Convergence epoch tradeoff of MLP. Epochs limit= 100

(d) Loss-Convergence epoch tradeoff of RNN

Figure 2: Impact of data loss on convergence and JCT

ML jobs encounter a tail latency much higher than average flow completion time (FCT). As shown in table 1, compared to finishing all TCP flows within average FCT, the tail FCT increases the job completion time by 10% to 93%. (Detailed evaluation setup will be explained in §3.1)

2.2 Opportunities from bounded-loss tolerance

The tail flow completion time results from the cost of recovering lost packets. Instead of designing complex mechanism to recover data losses, we take one step back and ask: Can we directly *ignore* some data losses? To give a positive answer, we first go through observations that machine learning achieves same application level performance in presence of ignored packet losses, and discuss the intuition behind this phenomenon.

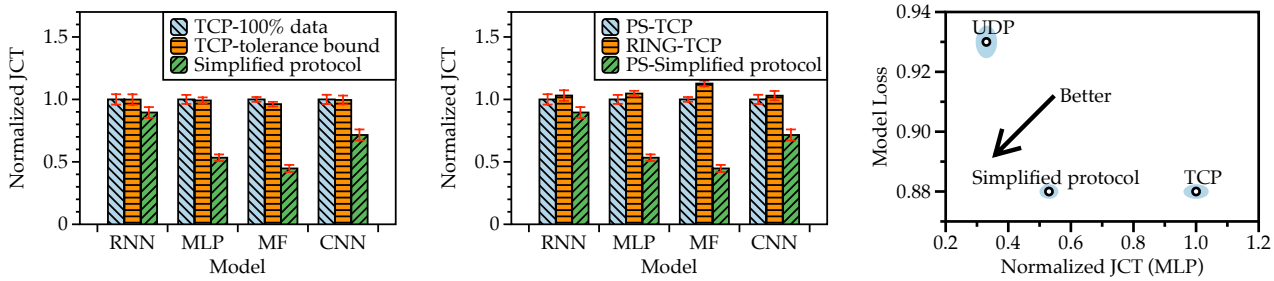
Empirical evidence shows that SGD tolerates bounded data loss. We simulated the scenario of distributed SGD over lossy network without retransmitting lost packets: We execute distributed training on 4 GPUs, within each iteration, a fraction of the gradients collected from each GPU is randomly marked as lost. The lost probability p start from 0, adds 5% each time until $p = 1$. The networking protocol is TCP when $p = 0$, and a bounded-loss tolerant protocol described in §3.1 in other cases. As shown in figure 2, for the two different models we select, data loss at certain fraction results shorter job completion time with same epochs (times to go through the dataset).

We describe this phenomenon as *bounded-loss tolerance feature of SGD*. Specifically, random data loss of different probability p has different impacts on the convergence of a ML job. There exists probability x_1, x_2 such that for all p:

- When $p \leq x_1$, job converges with same epochs, leading to shorter job completion time.
- When $x_1 < p \leq x_2$, job converges but with more epochs. In this range, larger p leads to longer job completion time with same performance.
- When $p > x_2$, job cannot converge to baseline value, i.e., job performance degrades.

As illustrated in figure 2, $(x_1, x_2) = (15\%, 55\%)$ for MLP and $(10\%, 90\%)$ for RNN, respectively. selecting $p \approx x_1$ gives the shortest job completion time, in that it tolerates highest packet drops without additional computation cost. We describe x_1 as *loss tolerance bound*.

Reason for bounded-loss tolerance: Machine learning algorithms such as SGD are inherently approximation algorithms, therefore delivering precise results in each iteration is not necessary. Due to its greedy loss-reduction strategy, SGD is capable of fixing the error resulting from certain data loss in later iterations. However, unbounded data loss might corrupt ML job's performance, as under too many data losses, SGD algorithms may not find sufficient information to derive accurate results. An extreme case would be ML models lose all intermediate data and SGD cannot make any progress.



(a) Marginal gain for message compression **(b) Similar JCT for PS & RING** **(c) Potential for bounded-loss tolerance**

Figure 3: Comparison of baseline TCP w/ different optimizations for distributed ML

Abbrv.	Model	Application
MF	Matrix Factorization [9]	Recommendation
MLP	Multi-layer Perceptron [21]	Text Classification
CNN	ResNet20 [11]	Image Classification
RNN	LSTM seq2seq [22]	Translation

Table 2: Models used for evaluations

Model	RNN	MLP	MF	CNN
Loss tolerance bound	0.10	0.15	0.35	0.15

Table 3: Different models tolerates different loss

Model	RNN1	RNN2	CNN1	CNN2
Loss tolerance bound	0.10	0.15	0.15	0.15

Table 4: Same model, different data tolerates similar loss

Finding tolerance bound for different model and data. We run the models in table 2 and analyzed the running time until convergence with different data loss rate. As shown in table 3, different models tolerates different data losses. Table 4 is conducted on two different datasets each on different models. We can see for the same model trained with different datasets, the loss tolerance bound is similar. Therefore, the loss tolerance bound can be estimated based on previous experience.

2.3 The need for new transport layer solution

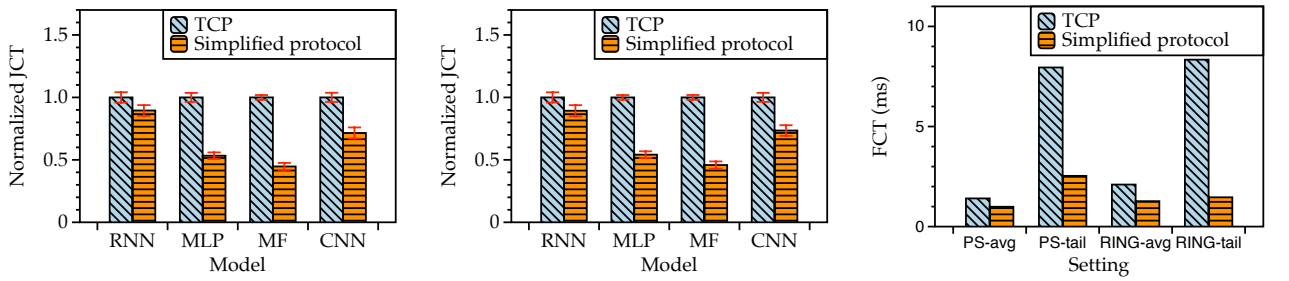
An intuitive method to leverage bounded-loss tolerance feature to accelerate distributed ML is to reduce the message size at sender side by randomly dropping parameters. However, this application-level solution only leads to marginal gain, as the performance of transmitting reduced size messages are still limited by the long tail transport layer protocols. Other application level treatments and existing transport layer protocols also doesn't solve this problem. This motivates us seek for new transport level solutions.

Why not dropping parameters directly? One way to optimize distributed ML with bounded-loss tolerance feature is to randomly drop a fraction p of data and transfer the remaining. However, as the reduced data are still sent through a reliable protocol, the cost of packet loss and recovery cannot be avoided. Figure (3a) illustrates this impairment. We reduce the parameter size transferred in the application level to the model's loss tolerance bound, sending reduced data with TCP has similar job completion time compared to sending all data with TCP. The potential of improvement is much smaller than using a transport layer solution.

Why not better parameter synchronization scheme? Different parameter synchronization schemes are proposed for better utilizing the network [16, 20]. These schemes still depend on transport protocol's performance and are also prone to long tail latency. As figure (3b) shows, using different flow scheduling optimizations has comparable performance in job completion time, and potential performance gain is much smaller than transport layer solutions.

Why not existing transport protocols? Existing RDT protocols are prone to the cost of loss recovery (e.g. TCP timeouts). On the other hand, Unreliable Data Transfer (UDT) protocols have no guarantee on the fraction of data received, in the worst case the data loss rate may significantly exceed the loss tolerance bound, therefore UDT protocols are not suitable for distributed ML jobs.

Figure (3c) illustrates the impairments of existing protocols, with TCP and UDP as examples. For TCP, guaranteeing the delivery of every packet results in large tail completion time. For UDP, A high rate may result in dropping more packets than application can tolerate, degrading the application level performance; Ideally, by cutting the tail latency of TCP flows, ML jobs can complete with same application performance, while achieving a smaller job completion time resulting from reduced communication time.



(a) Speedup under PS scheme **(b) Speedup under RING scheme** **(c) FCT of simplified protocol and TCP (model:MLP)**

Figure 4: Speedup of simplified protocol against TCP

3 TOWARDS BOUNDED-LOSS TOLERANT PROTOCOL

In this section, we discuss the potential of a bounded-loss tolerant protocol. The bounded-loss tolerant protocol should guarantee the delivery of a predefined fraction of data while delivering the rest of the data in a best effort manner. The protocol should handle packet losses as follows:

- Packet losses that can be quickly detected (e.g., duplicated ACKs) will be retransmitted as original.
- *Bounded* packet losses with long detection time (e.g., retransmission timer) can be directly discarded.

In this section, we demonstrate the advantages of an simplified protocol over existing protocols. We then discuss some challenges of integrating a bounded-loss tolerant protocols with ML application, and the intuition why it is tractable.

3.1 Potential improvements

Bounded-loss tolerance feature opens new avenues to congestion control design. Below a certain loss rate, unrecovered packet drops will not affect ML application performance. Since message size is purely determined by the ML model, in theory receivers are capable of deciding when the received data is enough to guarantee application level performance.

We quantify the full potential of a bounded-loss tolerant protocol using an *simplified* version. The simplified protocol has no packet ordering requirements and only counts received data amount. The simplified protocol sends all data at sender side, enables fast retransmission mechanisms, and we measure the time when a predefined fraction p of total data is received. The tolerance bound p follows table 3.

Experimental settings. We simulated an oversubscribed spine-leaf topology with 4 spine switches, 8 leaf switches and 16 servers in each rack. Within the same rack 8 servers generate distributed machine learning traffic, the other servers generate background traffic for other applications. These applications create a load factor of 0.5 on core links. The over subscription ratio is 4:1. All the links in the topology is 40Gbps link. The switch buffer is configured of size 450KB, with an ECN marking threshold of 90KB. Default protocol is DCTCP with initial window of 10 packets. We generate realistic workloads of data center same as [5] and employ Equal Cost Multi Path (ECMP) for load balancing.

Abbrv.	Dataset	Batch Size	Learning Rate
MF	[10]	256	0.1
MLP	[2]	32	0.5
CNN	[15]	32	0.1
RNN	[1]	32	0.25

Table 5: Detailed models for evaluation

The detailed hyper parameter settings can be found in table 5. We use fixed learning rate without weight decay and momentum factors. The convergence of ML jobs are defined to be the validation loss (i.e. model loss on a subset of data not used for training) decreases to a model-specific threshold. Each model is trained for 100 epochs.

Estimating job completion time. We run the ML jobs simulating packet drops as §2.2 until the loss value is smaller than baseline (ML jobs without data losses) and epochs passed is no smaller than baseline. The simulated job completion time is then

$$\#\text{ epochs} \times (\text{Compute time} + 2 \times \text{Network time}) \quad (1)$$

We run a simulated ML jobs offline with real data, while emulating the data drops for each parameter uniformly at random. The random data loss rate is set as tolerance bound in table 3.

Preliminary results. Figure 4 shows the improvements against TCP when we run the models in table 2. We compare the performance of the simplified protocol with the

performance of TCP. We find that leveraging the loss tolerant feature of SGD can greatly benefit distributed ML jobs. For the models we select, the simplified protocol brings 1.1x-2.2x speedups. The results of different parameter schemes are very similar. Due to the difference in the time spent in computation and communication in each model, we see different speedups for different models. Figure 3c shows that the simplified protocol gets faster speed than TCP and provides a better application level performance than UDP.

In figure (4c), we plot compare the flow completion time of all the flows generated by the MLP jobs. As figure shows, the main benefit of the simplified protocol is to cut the tail latency. The average FCT is similar with that of TCP. Therefore, the simplified protocol is effective in cutting the tail completion time of ML message transfers.

3.2 Challenges

Implementing a loss tolerant protocol is faced with an important challenge. As the protocol only specifies the highest loss rate tolerated, the received packet contents may contain multiple fragments that of the original message. In other words, the server may have received enough message in terms of size, but how to interpret the message remains a problem.

Is it possible to understand the partially received message? The answer is positive. The key observation is that machine learning messages can be abstracted as a vector describing the model. The vector contains floating point numbers of 32 bits or smaller size. To interpret packets' content independently, it is sufficient to restrict that the same floating point number is not segmented into two packets. For example, a packet with 1400 bytes of payload size is capable to contain 350 32-bit floating point numbers. In such cases, switching a on-the-boundary value to next packet reduces the packet size by less than 0.5%, meaning the additional packets generated by this requirement is negligible.

Idea #1: Range indexing of packets. Following the vector abstraction of ML models, each packet contains two integer values as ranged keys indicating the index range of the packet's content in the vector. The overhead of inserting ranged key index is low: With a normal packet payload size of ~ 1400 bytes, adding two 32-bit numbers as ranged index increases the overhead of less than 1%, an negligible overhead to network traffic.

Idea #2: Integrating with ML frameworks. Upon receiving enough packets, the bounded loss tolerant should forward the received message to upper layer application, i.e., the ML frameworks. In addition, the transport protocol marks the index ranges that the data is not received. This gives the ML application the information required to take average of received gradients on server side, or to use an earlier version of parameter value on worker side.

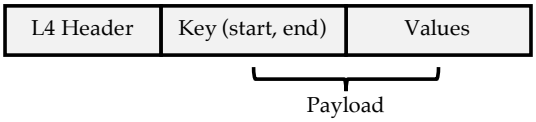


Figure 5: A ranged indexing scheme for packets.

4 RELATED WORKS

Optimizing distributed ML. There is a large body of work that optimizes distributed ML based on its stochastic features. Gradient compression [17, 23] and quantization [4, 24] transfer less accurate or compressed gradients. While these works greatly reduced the network load, their performance is limited by the transport protocol used. In high-speed data center network, this can lead to tail latency problems, degrading job performance.

Some works are proposed to improve distributed ML performance by removing network bottlenecks with less strict synchronization requirements [19, 25]. These methods do not have generalized good application level performance on all ML models, as discussed in [12]. In distributed training, parameters are often required to be synchronized in bulk synchronous parallel (BSP) to guarantee performance, i.e., every worker must finish syncing the parameters for current iteration before the next iteration starts.

Reducing tail latency. Several research works in data center network area are proposed to reduce the tail latency. Pfabric [6] and Cutting Payload [8] optimizes tail latency with fast detection of lost packets. While these methods are effective, they require changes of switch hardware. TCP based schemes like DCTCP [5] generally improve latency, but they have no guarantee on the worst case performance.

Loss tolerance traffics. Some protocols for real-time streaming applications tolerates packet loss, e.g., RTCP [13]. However, related applications doesn't have a strong tolerance bound as ML application does. It remains being explored whether a dynamic loss tolerance similar to quality-of-service (QoS) concept is applicable to distributed machine learning jobs.

5 CONCLUSION

In this paper, we call upon this community to develop transport layer solutions for distributed machine learning due to their stringent network requirements. Distributed machine learning jobs generate bursty traffic when synchronizing parameters, leading to long tail flow completion time and increase job completion time. We address the bounded-loss tolerance feature of machine learning algorithms, and propose to solve the tail completion time by safely ignoring some packet losses. Preliminary results show non-trivial performance gain on various distributed ML applications.

REFERENCES

[1] 2019. Lots of neat sentence pairs datasets. http://www.manythings.org/anki/

[2] 2019. Reuters newswire topics classification. https://keras.rstudio.com/reference/dataset_reuters.html

[3] Martín Abadi, Paul Barham, Jianmin Chen, Zhifeng Chen, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Geoffrey Irving, Michael Isard, et al. 2016. Tensorflow: A system for large-scale machine learning. In *OSDI*.

[4] Dan Alistarh, Demjan Grubic, Jerry Li, Ryota Tomioka, and Milan Vojnovic. 2017. QSGD: Communication-efficient SGD via gradient quantization and encoding. In *NIPS*.

[5] Mohammad Alizadeh, Albert Greenberg, David A Maltz, Jitendra Padhye, Parveen Patel, Balaji Prabhakar, Sudipta Sengupta, and Murari Sridharan. 2010. Data center tcp (dctcp). In *SIGCOMM*.

[6] Mohammad Alizadeh, Shuang Yang, Milad Sharif, Sachin Katti, Nick McKeown, Balaji Prabhakar, and Scott Shenker. 2013. pfabric: Minimal near-optimal datacenter transport. In *SIGCOMM*. ACM.

[7] Léon Bottou. 2010. Large-scale machine learning with stochastic gradient descent. In *COMPSTAT'2010*. Springer.

[8] Peng Cheng, Fengyuan Ren, Ran Shu, and Chuang Lin. 2014. Catch the whole lot in an action: Rapid precise packet loss notification in data center. In *NSDI*.

[9] Rainer Gemulla, Erik Nijkamp, Peter J Haas, and Yannis Sismanis. 2011. Large-scale matrix factorization with distributed stochastic gradient descent. In *KDD*.

[10] Grouplens. 2019. Movielens dataset. https://grouplens.org/datasets/movielens/

[11] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. 2016. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*.

[12] Kevin Hsieh, Aaron Harlap, Nandita Vijaykumar, Dimitris Konomis, Gregory R Ganger, Phillip B Gibbons, and Onur Mutlu. 2017. Gaia: Geo-Distributed Machine Learning Approaching LAN Speeds.. In *NSDI*.

[13] C Huitema. 2003. RFC 3605, Real Time Control Protocol (RTCP) attribute in Session Description Protocol (SDP). *Microsoft, Oct* (2003).

[14] Diederik P Kingma and Jimmy Ba. 2014. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980* (2014).

[15] Alex Krizhevsky, Vinod Nair, and Geoffrey Hinton. 2014. The CIFAR-10 dataset. online:http://www.cs.toronto.edu/kriz/cifar.html

[16] Mu Li, David G Andersen, Jun Woo Park, Alexander J Smola, Amr Ahmed, Vanja Josifovski, James Long, Eugene J Shekita, and Bor-Yiing Su. 2014. Scaling distributed machine learning with the parameter server. In *OSDI*.

[17] Yujun Lin, Song Han, Huizi Mao, Yu Wang, and William J Dally. 2017. Deep gradient compression: Reducing the communication bandwidth for distributed training. *arXiv preprint arXiv:1712.01887* (2017).

[18] Ning Qian. 1999. On the momentum term in gradient descent learning algorithms. *Neural networks* (1999).

[19] Benjamin Recht, Christopher Re, Stephen Wright, and Feng Niu. 2011. Hogwild: A lock-free approach to parallelizing stochastic gradient descent. In *NIPS*.

[20] Alexander Sergeev and Mike Del Balso. 2018. Horovod: fast and easy distributed deep learning in TensorFlow. *arXiv preprint arXiv:1802.05799* (2018).

[21] David Allister Simanjuntak, Heru Purnomo Ipung, Anto Satriyo Nugroho, et al. 2010. Text classification techniques used to faciliate cyber terrorism investigation. In *ACT*.

[22] Ilya Sutskever, Oriol Vinyals, and Quoc V Le. 2014. Sequence to sequence learning with neural networks. In *NIPS*.

[23] Jianqiao Wangni, Jialei Wang, Ji Liu, and Tong Zhang. 2018. Gradient sparsification for communication-efficient distributed optimization. In *NIPS*.

[24] Wei Wen, Cong Xu, Feng Yan, Chunpeng Wu, Yandan Wang, Yiran Chen, and Hai Li. 2017. Terngrad: Ternary gradients to reduce communication in distributed deep learning. In *NIPS*.

[25] Eric P Xing, Qirong Ho, Wei Dai, Jin Kyu Kim, Jinliang Wei, Seunghak Lee, Xun Zheng, Pengtao Xie, Abhimanu Kumar, and Yaoliang Yu. 2015. Petuum: A new platform for distributed machine learning on big data. *IEEE Transactions on Big Data* (2015).

[26] David Zats, Tathagata Das, Prashanth Mohan, Dhruba Borthakur, and Randy Katz. 2012. DeTail: reducing the flow completion time tail in datacenter networks. In *SIGCOMM*. ACM.