Towards Alleviating Bottlenecks in Distributed Training of Deep Neural Nets (DNN)

Muhammad U. Hashmi
usama@distreesync.com

Abstract

Training Deep Neural nets (DNNs) is a computationally extensive task which require immense compute power in the form of CPUs and GPUs. And because of decaying moores a single node can only provide limited compute power which is insufficient to quickly train modern DNNs. This gives rises to distributed training of DNNs over a cluster of GPU enabled nodes. However distributed training of DNN generates huge network overhead traffic which becomes a bottle neck in reducing total training duration time. In this paper we focus on reducing the network over head to reduce the training time of DNNs.

We are proposing a FPGA based hardware accelerator approach to reduce the network overhead traffic generated during distributed training of DNNs. The FPGA based hardware accelerator acts as a in-network switch which also performs part of the computation for training DNNs. We will evaluate the training duration of DNNs using hardware accelerator on a cluster of 4 GPU enabled nodes. This is an on-going project and currently we are in process of building hardware accelerator.

1 Introduction

In recent years Deep Neural Nets (DNN) have evolved to solve complicated problems in the domain of computer vision [19], autonomous systems, stock predictions and natural language processing etc.

As machine learning models take on to solve more complicated problems their model size increases and they are trained over even bigger and diverse data sets. Training these huge multi-layered deep neural net models requires a lot of compute resources. For instance it would take two days to train ResNet-50 [9] for 90 epochs over ImageNet data set [7] using a single Tesla V100 GPU. And during development phase DNN models are trained multiple times for hyper-parameter optimization

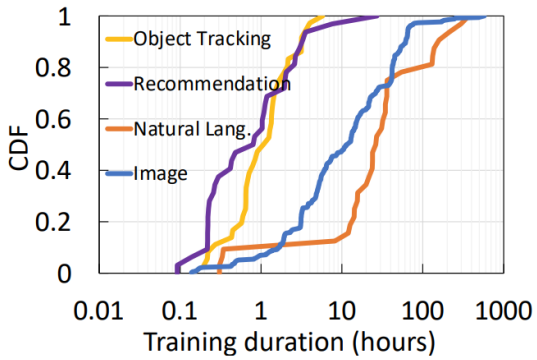

Figure 1: Training duration for multiple DNN models

and to improve performance which further elongates total training duration. The intrinsic approach to reduce the training time is to train the DNN model over a cluster of GPU enabled nodes instead of a single node.

However the problem persists even when we train DNN models over distributed cluster of GPU enabled nodes. Just to emphasis on the severity of the problem Figure 1 shows a cumulative distribution function plot for total training duration time in hours over a log scale for multiple DNN models. It shows that more than 50% of natural language processing models takes over a day to train. It turns the root cause of the long training duration is the excessive network overhead traffic generated while training DNNs over a distributed cluster. Hence the main problem we are tackling in this paper is to reduce the network overhead generated while training DNNs on a distributed cluster without compromising model accuracy. By reducing the network overhead will allow us to reduce the training time of DNNs from days or weeks to hours.

The rest of the paper is structured as follows. In section 2 We explain the background of distributed DNN training, possible fronts to reduce the network overhead generated during distributed DNN training (rel_work). In

section 3 we explained why our adapted approach is innovative and overcomes the limitations of prior work and its analysis with contemporary research work (approach and limitations it overcomes), In sec 4 we analyze our experimental findings.

2 Background and Related work

To clearly explain distributed DNN training I will briefly review the iterative gradient descent algorithm while training on a single node. As the exact steps of gradient descent algorithm is implemented in a parallel manner to train DNNs in a distributed manner.

Training a DNN on a single node follows three sequential steps. 1) The input data is used to create a prediction output from DNN model. And that prediction is compared with label of that particular input data to calculate error. This is referred as forward pass. 2) Using the prediction error value the gradient value is calculated for all the parameters in the model in a reverse order. This is referred as backward pass. 3) Finally each parameter value is updated according to its particular gradient value, scaled in proportion to learning rate parameter. The mathematical form of stochastic gradient update is as follows:

$$G(w_t) = \sum_{i=1}^{n} l(x_i, y_i, w_t) + \Omega(w_t) \tag{1}$$

$$G(w_t) = \nabla L_t(w_t; \xi_t) \tag{2}$$

$$w_{t+1} = w_t - \eta G_t(w_t) \tag{3}$$

where $w_t \in \mathbb{R}$ is N-dimensional model parameter at t iteration, η is the learning rate and ξ_t is randomly sampled batch of data.

Now we will go through the prior work done to reduce network overhead generated while training DNNs on a cluster. Prior research work is categorized based on the approach they adapted, listed as: 1) to reduce the amount of data transmitted between nodes, 2) off-load part of the computation to network switch 3) use compression to reduce the data transmitted 5) increase the network speed.

2.1 Reduce Amount of Data Transmitted

Consider a situation where we are training a DNN model with one billion parameters over a cluster of 100 nodes using gradient descent approach mentioned earlier. Each node will perform forward pass and back propagation operations to update its distinct model parameters. Now each node needs to update its in proportion to the updated parameters of other 999 nodes. This sharing of parameters would result in a network traffic of 1 Tera-bytes per iteration. We will look at multiple approaches to reduce

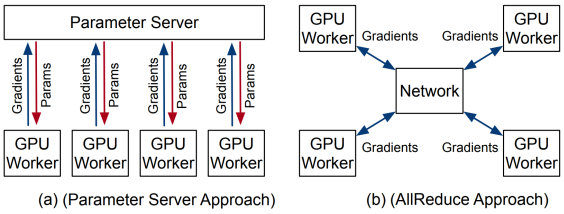

Figure 2: (a) Parameter server (PS) is based on master slave architecture where PS node and worker node interact using push pull message transfer. (b) Allreduce is base on peer-2-peer decentralized architecture. [14]

this excess network overhead. In order to make the most use of available cluster resources, a copy of DNN model is placed on each worker node and input data is iterated in the form of mini-batches. This approach is referred as data-parallel strategy. To implement data parallel strategy there are two widely implemented methodologies 1) Parameter Server approach [14]. 2) AllReduce approach.

In parameter server (PS) approach one of the node is designated as parameter server (PS) node while the rest are designated as worker nodes. Parameter server is built on centralized approach and is widely used in distributed training [15, 6, 10, 16, 18]. Parameter server is based on distributed sub gradient descent algorithm mentioned in figure 3. Parameter server implementation a copy of DNN model is placed on all nodes including PS node. The input data set is distributed over all worker nodes and they performs forward pass operation to obtain their respective gradient values. Afterwards each worker pushes its float gradient value to PS node where PS node performs a normalized aggregation on all gradient values received and performs back propagation operation and updates its distinct model parameters. Afterwards, the updated model parameters at the PS node are pulled by all worker nodes. The PS node maintains a centralized synchronization among all other nodes of network cluster, that is until the PS node has not completed back propagation operation and updated all of its parameter values all other nodes remain in halt mode before pulling updated parameter values from PS node.

Allreduce approach is a decentralized implementation of data parallel strategy. It follows the same algorithm with a slight modification. In Allreduce approach there is no PS node therefore once each node calculates gradient value by performing forward pass operation. Each node shares its gradient and acquires updated parameters from other nodes. It is important to mention that because of collective communication fashion of Allreduce it is not suitable for asynchronous communication. However Allreduce approach is used in applications [4, 5, 21, 11, 8] where it is necessary to avoid cen-

tralized servers.

Data parallel strategy reduces the amount of data transmitted between nodes by sharing only the gradient values from each node. However data parallel approach is built on the assumption that each node in the cluster has sufficient GPU memory that it can hold entire DNN model.

2.2 Offload Computation to In-network Switches

Researchers have tried to reduce training time of DNNs using high throughput in-network switches and network accelarators like Intel's FlexPipe [2], Cavium's XPliant [3], Barefoot Tofino [1]. Recently published works like SwitchML [20] and ATP [13] proposes to reduce network overhead by performing part of the gradient aggregation computation inside network switch while the data is transmitted between nodes. These in-network switches reduces the network overhead by reducing the confirmation and packet recovery protocols implemented in TCP data plane.

2.3 Compression

Researchers working on compression techniques have also published their work [17] about reducing network overhead traffic generated during distributed training of DNNs. However this is an orthogonal approach from our approach of computational network systems.

2.4 Increase Network Bandwidth

Another team of researchers working on photonic devices proposed a solution [12] where they used high throughput silicon photonics based network adapters to increase the bandwidth of network cluster and trained DNNs on it. However this is also an orthogonal approach.

3 Our Approach and Prior Limitations it overcomes

Our proposal is based on the idea to off-load part of the computation for training machine learning models to in-network switch. For that we should look in detail about the limitations associated with prior published work like SwitchML [20] and ATP [13].

SwitchML is the premier work that proposed to offload part of the distributed training computation to high throughput in-network switch. It implemented in-network aggregation on a programmable top of the rack (TOR) switch and reported to increase training throughput by 2X for VGG16 DNN. However because of the

Algorithm 1 Distributed Subgradient Descent

Task Scheduler:

issue LoadData() to all workers
for iteration $t = 0, \ldots, T$ **do**
    issue WORKERITERATE(t) to all workers.
end for

Worker $r = 1, \ldots, m$:

function LOADDATA()
    load a part of training data $\{y_{i_k}, x_{i_k}\}_{k=1}^{n_r}$
    pull the working set $w_r^{(0)}$ from servers
end function
function WORKERITERATE(t)
    gradient $g_r^{(t)} \leftarrow \sum_{k=1}^{n_r} \partial \ell(x_{i_k}, y_{i_k}, w_r^{(t)})$
    push $g_r^{(t)}$ to servers
    pull $w_r^{(t+1)}$ from servers
end function

Servers:

function SERVERITERATE(t)
    aggregate $g^{(t)} \leftarrow \sum_{r=1}^{m} g_r^{(t)}$
    $w^{(t+1)} \leftarrow w^{(t)} - \eta \left(g^{(t)} + \partial \Omega(w^{(t)})\right)$
end function

Figure 3: Distributed Subgradient Descent algorithm [14]

architecture of SwitchML it was implemented at a rack scale. This limits the number of worker nodes that can be used to train DNN modes.

Recently published ATP [13] paper proposed to scale distributed DNN training job not only to multiple racks which increased the number of worker nodes, it also scaled it to multi-tenant training jobs. This allows ATP to handle multiple distributed neural net training jobs in a decentralized manner.

Both ATP and SwitchML implemented a variation of distributed sub gradient descent algorithm mentioned in Figure 3 because it provides a synchronized methodology to train distributed neural nets. However as of recent time, the in-network switches used by SwitchML and ATP lacks 32- bit float point arithmetic support. And gradient values are represented using 32-bit float point type [13](Section 3.8). Both of these papers over came this obstacle by scaling the float point gradient value on integer scale and pushing it to network switch where network switch performs best effort aggregation. And finally when gradient values from all worker nodes are received the workers pulls the aggregated integer value and scales it back to float point using a predefined scaling factor. Even though ATP paper mentioned a detailed justification of their choice of scaling factor. Still they acknowledged that a large scaling factor could lead to overflow of aggregated gradients. The issue with using a scaling factor for converting float to integer data type

is the loss of accuracy. And updating weight values with a closely approximate value could seriously impact the accuracy of Deep neural net trained. As even a minor variation in gradient values could result in an exponential change for modern deep neural nets which are composed of multiple layers.

We propose a FPGA based hardware accelerator that implements distributed sub-gradient descent algorithm and support 32-bit float point arithmetic's on network data plane. We selected FPGA based hardware accelerator because of its rapid prototyping and availability of network and 32-bit float arithmetic libraries. However there is a valid concern that our hardware accelerator might not be optimized to operate at equivalent throughput of barefoot tofino [1] in-network switches. But we are aiming that our hardware accelerator would maintain computational accuracy while training deep neural nets on a distributed cluster. In our proposed implementation we will implement parameter sharing from PS node to worker nodes rather then broadcasting all parameter values from PS node to other worker nodes.

4 Results and Analysis

If we closely evaluate distributed sub gradient descent algorithm as explained in section 2.1. We realized that worker nodes remain in no operation state while PS node is performing back propagation and updating its parameter values. This intrigued us to verify if PS node is causing a straggler condition and to estimate the utilization duration of GPU resources.

We trained ResNet-50 over a cluster of four nodes equiped with Nvidia Titan X with 12 gb of GPU memory. We implemented distributed sub gradient descent algorithm, where one of the node is assigned as parameter server (PS) and remaining three nodes were assigned as worker nodes. As we are still in process of building FPGA based hardware accelerator that would support in-network aggregation. Therefore, all of the four nodes were connected via a 10 GB/s standard switch. We recorded the GPU memory utilization over the scale of time as shown in Figure 4.

Figure 4 shows a periodic memory utilization pattern which is linked with each iteration of DNN training. We also noticed the duration for low memory usage is higher then duration for high memory usage which is evident that each node waits longer for receiving updated parameters. We also observed that a certain minimum amount of memory is always in use, this could be because of caching implemented for GPU operation. Upon comparison of memory utilization logs we were not able to verify if there is a straggler condition while PS node is performing back propagation and updating all parameters of its model. Possibly we would need to implement

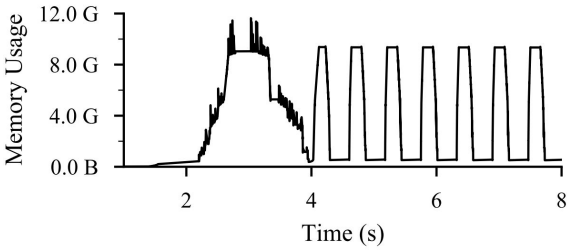

Figure 4: GPU memory utilization trace for training ResNet-50

a vector clock to verify if there is a straggler condition between PS and worker nodes.

5 Conclusion

In our problem statement we are focused on reducing the network overhead generated while training neural nets over a GPU enabled distributed cluster. We propose to over come the accuracy limitations associated with prior in-network implementations for training deep neural nets (DNN) by replacing the programmable network switch with FPGA based hardware accelerator.

References

[1] Barefoot tofino. *https://www.barefootnetworks.com/technology/tofino.*.

[2] Intel flexpipe. *https://www.intel.com/content/dam/www/public/us/en/documents/product-briefs/ ethernet-switch-fm6000-series-brief.pdf*.

[3] Xpliant ethernet switch product family. *http://www.cavium.com/XPliant-Ethernet-Switch-Product-Family. html.*.

[4] AWAN, A. A., HAMIDOUCHE, K., HASHMI, J. M., AND PANDA, D. K. S-caffe: Co-designing mpi runtimes and caffe for scalable deep learning on modern gpu clusters. In *Proceedings of the 22nd ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming* (2017), pp. 193–205.

[5] CHU, C.-H., LU, X., AWAN, A. A., SUBRAMONI, H., HASHMI, J., ELTON, B., AND PANDA, D. K. Efficient and scalable multi-source streaming broadcast on gpu clusters for deep learning. In *2017 46th International Conference on Parallel Processing (ICPP)* (2017), IEEE, pp. 161–170.

[6] DEAN, J., CORRADO, G., MONGA, R., CHEN, K., DEVIN, M., MAO, M., RANZATO, M., SENIOR, A., TUCKER, P., YANG, K., ET AL. Large scale distributed deep networks. *Advances in neural information processing systems 25* (2012).

[7] DENG, J., DONG, W., SOCHER, R., LI, L.-J., LI, K., AND FEI-FEI, L. Imagenet: A large-scale hierarchical image database. In *2009 IEEE conference on computer vision and pattern recognition* (2009), Ieee, pp. 248–255.

[8] GOYAL, P., DOLLÁR, P., GIRSHICK, R., NOORDHUIS, P., WESOLOWSKI, L., KYROLA, A., TULLOCH, A., JIA, Y., AND HE, K. Accurate, large minibatch sgd: Training imagenet in 1 hour. *arXiv preprint arXiv:1706.02677* (2017).

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

[10] Ho, Q., Cipar, J., Cui, H., Lee, S., Kim, J. K., Gibbons, P. B., Gibson, G. A., Ganger, G., and Xing, E. P. More effective distributed ml via a stale synchronous parallel parameter server. *Advances in neural information processing systems 26* (2013).

[11] Jia, X., Song, S., He, W., Wang, Y., Rong, H., Zhou, F., Xie, L., Guo, Z., Yang, Y., Yu, L., et al. Highly scalable deep learning training system with mixed-precision: Training imagenet in four minutes. *arXiv preprint arXiv:1807.11205* (2018).

[12] Khani, M., Ghobadi, M., Alizadeh, M., Zhu, Z., Glick, M., Bergman, K., Vahdat, A., Klenk, B., and Ebrahimi, E. Sip-ml: high-bandwidth optical network interconnects for machine learning training. In *Proceedings of the 2021 ACM SIGCOMM 2021 Conference* (2021), pp. 657–675.

[13] Lao, C., Le, Y., Mahajan, K., Chen, Y., Wu, W., Akella, A., and Swift, M. {ATP}: In-network aggregation for multi-tenant learning. In *18th USENIX Symposium on Networked Systems Design and Implementation (NSDI 21)* (2021), pp. 741–761.

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

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

[16] Li, M., Andersen, D. G., Smola, A. J., and Yu, K. Communication efficient distributed machine learning with the parameter server. *Advances in Neural Information Processing Systems 27* (2014).

[17] Lin, Y., Han, S., Mao, H., Wang, Y., and Dally, W. J. Deep gradient compression: Reducing the communication bandwidth for distributed training. *arXiv preprint arXiv:1712.01887* (2017).

[18] Ooi, B. C., Tan, K.-L., Wang, S., Wang, W., Cai, Q., Chen, G., Gao, J., Luo, Z., Tung, A. K., Wang, Y., et al. Singa: A distributed deep learning platform. In *Proceedings of the 23rd ACM international conference on Multimedia* (2015), pp. 685–688.

[19] Redmon, J., Divvala, S., Girshick, R., and Farhadi, A. You only look once: Unified, real-time object detection. In *Proceedings of the IEEE conference on computer vision and pattern recognition* (2016), pp. 779–788.

[20] Sapio, A., Canini, M., Ho, C.-Y., Nelson, J., Kalnis, P., Kim, C., Krishnamurthy, A., Moshref, M., Ports, D. R., and Richtárik, P. Scaling distributed machine learning with in-network aggregation. *arXiv preprint arXiv:1903.06701* (2019).

[21] Wang, S., Li, D., Geng, J., Gu, Y., and Cheng, Y. Impact of network topology on the performance of dml: Theoretical analysis and practical factors. In *IEEE INFOCOM 2019-IEEE conference on computer communications* (2019), IEEE, pp. 1729–1737.