Asynchronous Distributed Data Parallelism for Machine Learning

Zheng Yan, Yunfeng Shao
Shannon Lab, Huawei Technologies Co., Ltd.
Beijing, China, 100085
{yanzheng, shaoyunfeng}@huawei.com

Abstract

Distributed machine learning has gained much attention due to recent proliferation of large scale learning problems. Designing a high-performance framework poses many challenges and opportunities for system engineers. This paper presents a novel architecture for solving distributed learning problems in framework of data parallelism where model replicas are trained over multiple worker nodes. Worker nodes are grouped into worker groups which enable model replicas to be asynchronously aggregated via peer-to-peer communication. Merits of this framework include elastic scalability, fault tolerance, and efficient communication.

1 Introduction

The past decade witnessed the popularity and prosperity of machine learning, especially deep learning. In various applications such as speech recognition [1], image classification [2], object detection [3], face recognition [4], and natural language processing [5], deep learning has achieved state-of-the-art performances. However, this has come at a cost of huge computational burden. Many web applications require training data with TB/PB size. For example, WeChat, a mobile social platform has more than 400 million active users. If each user produces 1KB data, then the total data size would exceed 400TB. It is virtually intractable to store or process such a big data set on a single machine. As a result, a big data set is partitioned to subsets, assigned to multiple machines, and processed in a distributed manner. The benefits of distributed machine learning not only lie in the capabilities to deal with extremely large datasets, but also in the attainment of large models. It is observed in many deep learning applications that deeper and/or wider network architectures generally result in better performance [6].

Two main strategies are widely used for parallelizing data training, namely data parallelism and model parallelism. Data parallelism replicates multiple models over different partitioned subsets, and aggregates gradients for model update. Model parallelism divides a model to several parts and each part is processed on a worker node, which reduces memory consumption at a price of increasing communication cost. A hybrid use of the two strategies is also adopted in some applications [7]. Whatever strategies are applied for parallelism, the issue of scattering model parameters across different worker nodes is ubiquitous. The parameter server approach offers an efficient mechanism for model update, consistency, and synchronization [8]. The parameter server serves as a globally shared dictionary that is accessible by all worker nodes. In case of a group of parameters are used, each parameter server holds and updates a part of the model parameters.

Although significant progress has been made in distributed machine learning, many issues such as scalability, communication, and fault tolerance deserve further investigations. To facilitate efficient implementation of distributed machine learning, we propose an alternative data parallelism framework based on peer-to-peer communication. Unlike the parameter server mode, there is no globally shared dictionary in the cluster. Instead, asynchronous parameters scattering is performed with parallel instances of model replicas on each worker node. Moreover, worker nodes are grouped into worker groups and the number of worker groups can be dynamically adjusted. The proposed framework offers a high degree of fault tolerance as well as the elastic scalability.

2 Related work

The data explosion in recent years surged interests on distributed machine learning in both academia and industry. Many researchers explored scaling out machine learning algorithms through various parallelism methodologies. Generally speaking, existing efforts can be categorized into algorithmic paths and system paths. Algorithmic efforts mainly focus on the study of parallelizable optimization algorithms with the aim of faster convergence, reduced variance, and lower complexity [9, 10]. System efforts mainly focus on investigations on data storage, programming interface, and communication protocols. In framework of mandate synchronization and iterative communication, Mahout [11], based on Hadoop, and Mlib [12], based on Spark, offer two environments for creating scalable machine learning applications. However, the iterative communication over MapReduce [13] results in high communication cost. To alleviate this problem, Dean et al. proposed a framework called DistBelief where an asynchronous stochastic gradient decent algorithm, namely Downpour SGD, was applied for parallel training [14]. Each worker independently fetches parameters from a parameter server, computes the local gradient, and pushes it back to the server. The server immediately updates global model parameters using the received gradient information. GraphLab [15], a graph based framework, offers a parallel programming abstraction targeted for sparse iterative graph algorithms. One limitation is that the graph representation may not be efficient for some machine learning algorithms. Based on a bounded-asynchronous key-value store and a scheduler for iterative machine learning computations, Petuum [16] offers a general purpose platform to run machine learning applications at scale. Li et al. proposed a third-generation parameter server framework where both data and workloads are distributed over worker nodes [8]. Advantages of this framework include controllable asynchrony and user-definable communication reduction.

Advances in distributed learning frameworks proliferated implementations of related systems at Google [14], Facebook [17], Yahoo [18], and many other companies. Based on DistBelief, Google built a CPU cluster in light of hybrid data parallelism and model parallelism to train a deep network with over 1 billion parameters using 16000 CPU cores. In addition, Google developed COTS [19], an HPC system powered by GPU clusters with Infiniband interconnects and MPI. COTS is able to train 1 billion parameter networks with 3 GPU machines in several days. Facebooks multi-GPU cluster is able to train the AlexNet [20] using the ImageNet 1K data set by 4 Nvidia Titan GPUs in a couple of days. Baidu Paddle platform [21] applies a hybrid use of data parallelism and model parallelism based on GPU clusters, which support online services such as voice search and Baidu map. Mariana [7], a deep learning platform developed by Tencent, offers a multi-GPU data parallelism framework for DNNs, a multi-GPU model parallelism and data parallelism framework for deep CNNs, and a CPU cluster framework for large scale DNNs. These successful commercial applications manifest the blossom of distributed machine learning.

3 Architecture

The goal of this work is to develop a distributed machine learning framework that meets the following requirements. (1) Elastic scalability. Computing resources in a data center process multiple learning tasks simultaneously. It requires dynamic allocation of computing resources depending on the characteristics of the upcoming task. Elastic scalability aims to add or remove worker nodes without causing catastrophic failures of the training. (2) Fault tolerance. Node failures are inevitable in practice. Fault tolerance aims to continue the training even if node failure happens. (3) Efficient communication. A major bottleneck of distributed learning lies in the communication overhead as frequent parameter swapping across worker nodes is required. Efficient communication aims to reduce network traffic overhead, as well as improving utilization of network bandwidth. (4) Flexibility. A variety of machine learning toolkits such as Caffe and Torch are widely used in the community. Flexibility aims to enable users to make choices based on their preferences. In addition, experienced users may desire the flexibility of balancing the algorithmic convergence rate and system efficiency.

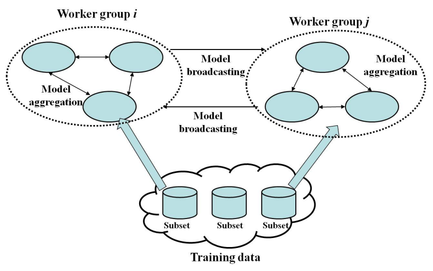

Figure 1: Overall architecture.

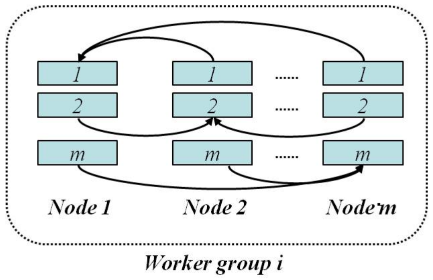

Figure 2: Communication protocol within a workder group.

The overall architecture of the proposed framework is illustrated in Fig. 1. Denote X as a training data set with finite number of samples. Let p be the number of worker groups where each group consists of a few worker nodes. The number of worker nodes in each worker group needs not necessarily be the same. Let N be the total number of worker nodes. The training set is partitioned into N subsets, and each subset is assigned to a worker node for local computation. Distributed learning based on the proposed framework has two main operations: model aggregation and model broadcasting.

Model aggregation: Assume a worker group i consists of m worker nodes, and each worker node stores a model replica denoted as $w_{i,j}, j = 1, \cdots, m$. Each worker node first computes local gradient $\Delta w_{i,j}$ of the objective function employed by the learning algorithm using locally stored subset of the training data. Then the worker node updates its local model $w_{i,j}$ based on the gradient, i.e., $w_{i,j} \leftarrow g(w_{i,j}, \Delta w_{i,j})$ where $g(\cdot)$ denotes an update rule. After k epochs of local learning (k is a user-defined integer), each worker node scatters its model $w_{i,j}$ to the other worker nodes in the same worker group. As a result, an aggregated model w_i is obtained that is virtually trained using m subsets. The way to perform parameter scattering is not unique. For example, it can be executed hierarchically following the widely used tree structure. We herein propose a more efficient approach to model aggregation where each model $w_{i,j}, j = 1, \cdots, m$, is splitted into m partitions and each worker node is responsible for aggregating 1 partition as shown in Fig. 2, where each worker node scatters $1/m$ model parameters to the other nodes within the same worker group.

Model broadcasting: Once the model aggregation is completed at the worker group i, it immediately broadcasts the updated model w_i to its neighboring worker groups. The receiving worker group j holds w_i in the buffer to wait for the completion of its own model aggregation after which w_j is replaced by w_i. As such, the worker group j continues local training in the following k epochs based on the model obtained from the worker group i. The aim of model broadcasting is to enable sufficient information exchange across worker nodes in an efficient way. It is worth noting that model broadcasting can be asynchronous.

The features of the proposed distributed machine learning framework are summarized as follows: multiple worker nodes are grouped into a worker group, and multiple worker groups asynchronously perform parameters swapping. The parameters swapping processes consist of two phases, namely model aggregation and model broadcasting. The model aggregation is performed within a worker group, whereas the model broadcasting is performed across worker groups. This framework results in a few contributions: First, it enables elastic scalability. Worker nodes can be dynamically added or removed without restarting the training by controlling the asynchrony. Secondly, it improves fault tolerance under the help of data repartition. If one worker node fails, its peers in the same worker group can dynamically take over its job. Even if a whole worker group fails, the other worker groups continue processing their training data and updating the model parameters. Thirdly, it offers a more efficient communication protocol. Considering the model aggregation with linear partition, the total data transfer flows are $\frac{2(m-1)}{m}$ of the model dimension. However, as most network technologies support duplex mode, the actual communication overhead is equivalent to $\frac{m-1}{m}$. For the model broadcasting operation, the communication overhead is a full model size. As a result, the total communication cost for a worker node is $1 + \frac{m-1}{m}$. On the other hand, the communication cost of a worker node in framework of parameter server is usually 2 as the fetch and push steps cannot be implemented simultaneously.

In physical view, it is clear that different algorithms perform varied on different computing architectures. For example, GPU is extremely powerful for algorithms based on floating point matrix operations, CPU is competent for rule-based learning, and FPGA is suitable for learning based on fixed-point matrix operation. The proposed peer-to-peer asynchronous communication supports various computing architectures. We even tested a hybrid use of multiple computing architectures to form a heterogeneous cluster. We implemented the proposed framework for providing machine learning cloud services. Preliminary results on ImageNet classification show that a quasi linear speedup is achieved.

4 Conclusion

In this paper we introduced a distributed data parallelism framework for machine learning applications. This framework offers a novel mechanism for model aggregation and parameter swapping based on peer-to-peer model communication. A linear partition strategy is developed for model scattering with the aim of reducing communication cost. Our implementation of the proposed framework in cloud service indicates that it provides fault tolerance, network efficiency and speedup to various machine learning applications

References

[1] Graves, A. & Jaitly, N. Towards end-to-end speech recognition with recurrent neural networks. In *ICML*, 2014.

[2] Szegedy, C., et al. Going deeper with convolutions. In *CVPR*, 2014.

[3] Girshick, R. Fast R-CNN. In *ICCV*, 2015.

[4] Taigman, Y., Yang, M., Ranzato, M. A., & Wolf, L. Deepface: Closing the gap to human-level performance in face verification. In *CVPR*, 2014.

[5] Hu, B., Lu, Z., Li, H., & Chen, Q. Convolutional neural network architectures for matching natural language sentences. In *NIPS*, 2014.

[6] Simonyan, K., & Zisserman, A. Very deep convolutional networks for large-scale image recognition. In *ICLR*, 2015.

[7] Zou, Y., Jin, X., Li, Y., Guo, Z., Wang, E., & Xiao, B. Mariana: Tencent deep learning platform and its applications. In VLDB, 2014.

[8] Li, M., et al. Scaling distributed machine learning with the parameter server. In *OSDI*, 2014.

[9] Agarwal, A. & Duchi, J. C. Distributed delayed stochastic optimization. In *NIPS*, 2011.

[10] Zhang, R. & Kwok, J. Asynchronous distributed admm for consensus optimization. In *ICML*, 2014.

[11] Apache Foundation. Mahout project, 2012. http://mahout.apache.org.

[12] Apache Foundation. Spark MLib, 2013. https://spark.apache.org/mllib/.

[13] Dean, J. & Ghemawat, S. MapReduce: simplified data processing on large clusters. *CACM*, 51(1):107113, 2008.

[14] Dean, J., et al. Large scale distributed deep networks. In *NIPS*, 2012.

[15] Low, Y., Bickson, D., Gonzalez, J., Guestrin, C., Kyrola, A., & Hellerstein, J. M. Distributed GraphLab: a framework for machine learning and data mining in the cloud. In *VLDB*, 2012.

[16] Ho, Q., et al. More effective distributed ML via a stale synchronous parallel parameter server. In *NIPS*, 2013.

[17] Yadan, O., Adams, K., Taigman, Y., & Ranzato, M. A. Multi-gpu training of convnets. *CoRR*, 2013.

[18] Ahmed, A., Aly, M., Gonzalez, J., Narayanamurthy, S., & Smola, A. J. Scalable inference in latent variable models. In *WSDM*, 2012.

[19] Coates, A., Huval, B., Wang, T., Wu, D. J., & Ng, A. Y. Deep learning with COTS HPC systems. In *ICML*, 2013.

[20] Krizhevsky, A., Sutskever, I., and & Hinton, G.E. ImageNet classification with deep convolutional neural networks. In *NIPS*, 2012.

[21] Mao, J., Xu, W., Yang, Y., Wang, J.,& Yuille, A. Deep captioning with multimodal recurrent neural networks (m-RNN). *ICLR*, 2015.