ADMMLIB: A Library of Communication-Efficient AD-ADMM for Distributed Machine Learning

Jinyang Xie, Yongmei Lei

ADMMLIB: A Library of Communication-Efficient AD-ADMM for Distributed Machine Learning*

Jinyang Xie and Yongmei Lei

School of Computer Engineering and Science, Shanghai University, Shanghai, China
{jyxie,lei}@shu.edu.cn

Abstract. Alternating direction method of multipliers (ADMM) has recently been identified as a compelling approach for solving large-scale machine learning problems in the cluster setting. To reduce the synchronization overhead in a distributed environment, asynchronous distributed ADMM (AD-ADMM) was proposed. However, due to the high communication overhead in the master-slave architecture, AD-ADMM still cannot scale well. To address this challenge, this paper proposes the ADMMLIB, a library of AD-ADMM for distributed machine learning. We employ a set of network optimization techniques. First, hierarchical communication architecture is utilized. Second, we integrate ring-based allreduce and mixed precision training into ADMMLIB to further effectively reduce the inter-node communication cost. Evaluation with large dataset demonstrates that ADMMLIB can achieve significant speed up, up to 2x, compared to the original AD-ADMM implementation, and the overall communication cost is reduced by 83%.

Keywords: Asynchronous ADMM · Consensus Optimization · Distributed Machine Learning.

1 Introduction

With the ever-increasing sizes of datasets and models, machine learning model training often takes so long time. Due to the single machine's limited computing resources, it is reasonable to distribute large scale machine learning workloads across multiple computing nodes. Distributed machine learning (ML) jobs often involve solving a non-convex, decomposable and regularized optimization problem. Distributed optimization is becoming a prerequisite for solving large scale ML problems. The alternating direction method of multipliers (ADMM) [1] is an optimization technique by decomposing the original problem into subproblems for parallel iterations. Usually, ADMM was implemented in master-slave architecture, in which a master coordinates the computation of a set of distributed

* Supported by the National Natural Science Foundation of China under grant No.U1811461.

workers. To reduce the synchronization overhead, recently, the synchronous distributed ADMM has been extended to the asynchronous setting [8, 2]. Asynchronous updates would improve the computation efficiency of the distributed ADMM.

A major performance bottleneck of AD-ADMM is the high communication overhead due to the following factors. First, large-scale ML are trending to learn large models with tens or hundreds of millions of parameters, generating a large amount of network traffic for distributed training. Second, under the master-slave architecture, a single incoming link to the master is shared across multiple workers, causing network congestion. Hence, it is important to reduce the communication overhead when scaling AD-ADMM to large-scale clusters.

In our work, we focus directly on the problem of improving the performance and scalability of the AD-ADMM. We employ a set of network optimization techniques, such as hierarchical communication architecture, ring-based allreduce and mixed precision training, to achieve load balancing and reduce communication overhead. We build a library named ADMMLIB integrating our optimization techniques. ADMMLIB manages details of parallelism, synchronization and communication. It provides simple programming interfaces for users to implement scalable AD-ADMM.

2 Related Work

Because of the demand for faster training of ML model, several frameworks have been proposed, such as Petuum [7]. The standard distribution strategy in ML is data parallelism. To implement the data-parallel model training, there are two design choices: the parameter server (PS) [3] approach using master-slave architecture and the ring-based allreduce [4] approach with P2P architecture. In PS, a logical parameter server aggregates model updates from all workers and broadcasts to all workers. One bottleneck of the PS is the high communication cost on the central server. In the ring-based allreduce, all the nodes are organized as a logical ring, and each node communicates with two of its peers. Ring-based allreduce is an algorithm with constant communication cost. Recent literature [5] has shown clear benefits of the ring-based allreduce. Distributed ADMM has been widely studied as an alternative method for distributed stochastic gradient descent algorithms. Recently [1] proved that the ADMM is suitable for distributed optimization problems. [8] has considered a version of asynchronous ADMM to speed up the ADMM. [2] added a penalty term based on [8] to improve the convergence efficiency of non-convex problems. [6] uses hierarchical communication structure to improve the communication efficiency of distributed ADMM.

3 ADMMLIB: System Design and Optimization

3.1 Hierarchical Communication Architecture

Although master-slave architecture has been widely used in the ADMM, it is not quite suitable for large scale machine learning. As shown in Fig. 1, ADMMLIB

adopts hierarchical communication architecture (HCA) to scale up to multicores on a single node, as well as scale out to multiple nodes in a cluster.

To scale up, ADMMLIB start a number of worker threads on each node and each worker thread is assigned to a dedicated CPU core. ADMMLIB also starts a coordinator thread on each node, and ADMMLIB will choose a coordinator as the master coordinator. Each worker only communicates with the coordinator on the same node. Coordinators communicate with each other to coordinate the computation of all workers in the cluster, therefore ADMMLIB can scale out to multiple nodes.

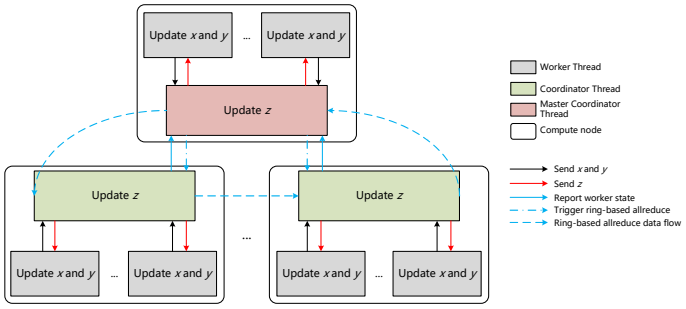

Fig. 1. Overview of ADMMLIB architecture.

Each worker i owns a train dataset partition and is responsible for updating x_i and y_i. x_i and y_i represent local model variable and dual variable [1], respectively. After sending up-to-date (x_i, y_i) to the coordinator on the same node, worker will block until they receive the updated z. z represents the global model variable [1]. Each coordinator takes charge of caching the latest (x_i, y_i) from workers. And each coordinator maintains its own copy of z. Replicas are kept consistent by exchanging data between coordinators. Specifically, when a coordinator receiving update from worker i, it reports to the master coordinator. Therefore, the master coordinator can know the status of all workers in the cluster. Once the partial barrier and bounded delay conditions [8] are satisfied, the master coordinator will inform all coordinators to perform an allreduce operation to update z.

Compared with the master-slave architecture, HCA can balance the load, because each coordinator (including the master coordinator) only needs to manage a small subset of workers.

3.2 Improvements on Internode Communication Strategies

The simple architecture of ADMMLIB also makes it easy to identify the limiting factors to scaling. Optimizing inter-node communication overhead is clearly the key to scaling.

Most of the inter-node communication overhead comes from the allreduce operation. Out of the possible allreduce implementation strategies, we choose the

ring-based allreduce algorithm. If the input data is m bytes, ring-based allreduce equally partition data into N_n chunks, N_n is the number of coordinators. Each coordinator sends and receives m/N_n bytes of data $2(N_n - 1)$ times to complete an allreduce operation. Thus, the total communication time is independent of the number of nodes. Ring-based allreduce distributes the communication cost across all N_n nodes to avoid a node becoming a performance bottleneck.

Model training is not very demanding for high-precision calculations. Compared to double-precision, using single-precision or even half-precision can increase arithmetic throughput without decreasing accuracy. Low precision training also helps to reduce communication overhead and memory storage requirement since the same number of values could be stored using fewer bits. We use mixed-precision training strategy in ADMMLIB. When optimizing x_i, y_i and z, ADMMLIB uses single-precision or double-precision parameters, depending on the user's choice. When caching and transferring parameters, ADMMLIB uses single precision to reduce memory usage and communication cost.

4 Experiment

In this section, we evaluate the performance and scaling efficiency of ADMMLIB. For comparison, we also use the multi-threading technique to implement the AD-ADMM in master-slave architecture, we call this system MAD-ADMM. We use a cluster of 5 computing nodes interconnected with a Gigabit Ethernet. Each node has two Intel E5-2690 CPU (2.9GHz/8core) processors and 64GB memory. In the experiment, we solve sparse logistic regression problem. We consider a large dataset: URL[1]. The URL has more than 2 million samples and 3 million features.

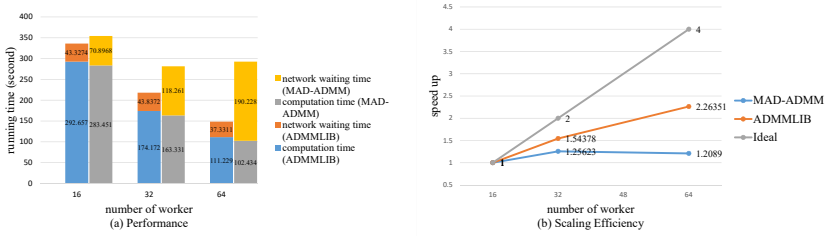

Fig. 2. Performance and scaling efficiency comparisons between ADMMLIB and MAD-ADMM.

We set up three experiments with 16, 32 and 64 workers, respectively. First, we test the performance of ADMMLIB. We compare two systems by running them to reach 20 iterations, and we recorded the computation time and network waiting time of the two systems. Fig. 2(a) shows the performance comparison. It

[1] https://www.csie.ntu.edu.tw/ cjlin/libsvmtools/datasets/binary.html#url

can be seen from Fig. 2(a) that as scaling increases the level of parallelism and (consequently) reduces the computation time of the two systems, the network waiting time of ADMMLIB changes little, however the network waiting time in the MAD-ADMM increases linearly. Therefore, ADMMLIB outperforms MAD-ADMM. ADMMLIB can reduce network waiting time by 62.9% when testing with 32 workers and reduce network waiting time by 83% when testing with 64 workers. Fig. 2(b) shows the scaling efficiency (taking the performance of 16 workers as the baseline). ADMMLIB has higher scaling efficiency thanks to the efficient ring-based allreduce algorithm. For 32 workers, we improved the scaling out efficiency from 62.8% to 77.1%. For 64 workers, we improved the scaling out efficiency from 30.2% to 57.5%.

5 Conclusion

Aiming at building a scalable and high-performance distributed model training system based on the AD-ADMM, this paper uses hierarchical communication architecture, ring-based allreduce algorithm and mixed precision training to reduce communication overhead and memory usage. Experiments show our system has higher performance and scalability than the original AD-ADMM implementation. But scalability of our system still does not reach ideal efficiency. In future work, we will try to optimize the sub-question solving efficiency to solve this problem.

References

1. Boyd, S., Parikh, N., Chu, E., Peleato, B., Eckstein, J., et al.: Distributed optimization and statistical learning via the alternating direction method of multipliers. Foundations and Trends® in Machine learning **3**(1), 1–122 (2011)
2. Chang, T.H., Hong, M., Liao, W.C., Wang, X.: Asynchronous distributed admm for large-scale optimization—part i: Algorithm and convergence analysis. IEEE Transactions on Signal Processing **64**(12), 3118–3130 (2016)
3. Li, M., Andersen, D.G., Park, J.W., Smola, A.J., Ahmed, A., Josifovski, V., Long, J., Shekita, E.J., Su, B.Y.: Scaling distributed machine learning with the parameter server. In: 11th {USENIX} Symposium on Operating Systems Design and Implementation ({OSDI} 14). pp. 583–598 (2014)
4. Patarasuk, P., Yuan, X.: Bandwidth optimal all-reduce algorithms for clusters of workstations. Journal of Parallel and Distributed Computing **69**(2), 117–124 (2009)
5. Sergeev, A., Del Balso, M.: Horovod: fast and easy distributed deep learning in tensorflow. arXiv preprint arXiv:1802.05799 (2018)
6. Wang, S., Lei, Y.: Fast communication structure for asynchronous distributed admm under unbalance process arrival pattern. In: International Conference on Artificial Neural Networks. pp. 362–371. Springer (2018)
7. Xing, E.P., Ho, Q., Dai, W., Kim, J.K., Wei, J., Lee, S., Zheng, X., Xie, P., Kumar, A., Yu, Y.: Petuum: A new platform for distributed machine learning on big data. IEEE Transactions on Big Data **1**(2), 49–67 (2015)
8. Zhang, R., Kwok, J.: Asynchronous distributed admm for consensus optimization. In: International Conference on Machine Learning. pp. 1701–1709 (2014)