PSMART: Parameter Server based Multiple Additive Regression Trees System

Jun Zhou[†], Qing Cui[‡], Xiaolong Li[†], Peilin Zhao[†], Shenquan Qu[‡], Jun Huang[‡]
[†]Ant Financial Group, [‡]Alibaba Cloud
Hangzhou, China
{jun.zhoujun, cuiqing.cq, xl.li, peilin.zpl, shenquan.qsq, huangjun.hj}@alibaba-inc.com

ABSTRACT

In this paper, we describe a Parameter Server based Multiple Additive Regression Trees system, or PSMART for short. Empirically, PSMART scales MART to hundreds of billions of samples and thousands of features in production.

Keywords

MART; LambdaMART; Parameter Server; XGBoost

1. INTRODUCTION

Nowadays, Machine Learning approaches have been extensively adopted by many Internet companies to extract information from Big Data, which can help them to make smart decisions to increase business output such as Gross Merchandise Volume [4]. Among the existing ML methods, MART and its extensions, such as LambdaMART, are widely used in industrial practices [3, 1], because of their high adaptability, practical effectiveness and strong interpretability. While the existing implementations of MART have been effectively applied to many generic cases with million level instances, it can not handle industrial datasets with hundreds of billions of samples and thousands of features [2]. In addition, the existing implementations can only be deployed in some independent ML environment, since they can not handle all kinds of failures in a complex production environment with many simultaneous tasks (e.g. Hive SQL, Map Reduce, Graph, MPI). To this end, in this work, we introduce PSMART, a Parameter Server (PS) based distributed learning system that reliably scales MART to support hundreds of billions of samples and thousands of features in the clusters. Our contributions are: (1) We built a general MART system based on PS which enjoys efficient sparse communication, better fault tolerance and ease of use (like the SQL style) etc; (2) We implemented a distributed LambdaMART with a query id based partitioning strategy. Due to these techniques, the system can scale near linearly across nodes, so that it is the fastest and the most scalable and stable MART implementation to the best of our knowledge.

WWW 2017 Companion, April 3–7, 2017, Perth, Australia.
ACM 978-1-4503-4914-7/17/04.
http://dx.doi.org/10.1145/3041021.3054225

2. RELATED WORK

Among all the existing work, XGBoost [2] is the most related to our PSMART. Specifically, XGBoost is a parallel tree boosting system based on Rabit[1], with approximate split-finding algorithm. However, it currently does not support distributed LambdaMART. In addition, Rabit only provides a limited fault tolerance mechanism and does not optimize the communication efficiency for sparse datasets. Our system can recover quickly because of the robust failover mechanism of PS. LightGBM [7] is another tool like XGBoost at present (parallel learning based on the socket or MPI, with a different regularization method). However, it also does not support failover and sparse communication.

3. PS BASED SMART

Parameter Server. Inside Alibaba, we have implemented a PS system [6, 8], which supports fault tolerance, and is deeply optimized for the communication efficiency of sparse data. On it, we developed PSMART, which is very robust due to its above advantages. For example, when some nodes fail, PSMART can automatically trigger the failover mechanism and recover from the nearest checkpoint.

MART on PS. To overcome the challenge of the huge storage need, PSMART employs a data parallelization mechanism. Specifically, each worker only stores a subset of the whole dataset for each feature. Under this mechanism, the main workflow for splitting a node is as follows: 1) Each worker calculates the local weighted quantile sketch for the data subset of the node stored on this worker; 2) Each worker pushes the local weighted quantile sketch to servers. The servers merge the local sketches to a global weighted quantile sketch by a multiway merge algorithm, and then use it to find the splitting value; 3) Each worker pulls the splitting value from servers and splits samples to two nodes. The details on producing local weighted quantile sketches and merging local sketches to a global sketch can be found at [2]. Repeating the above procedure can build a tree. In this way, we can build trees one by one following the gradient boosting framework to get a MART. Although the key idea is straightforward, there exists one key challenge: when the number of features gets bigger or the tree gets deeper, the computation and communication cost of split-finding algorithm will become very high. To remedy this issue, we leverage the communication schema of PS to reduce the cost of merging local sketches. This can further speed up the whole split-finding algorithm.

LambdaMART on PS. For LambdaMART, the procedure of building trees is the same with general MART al-

[1]https://github.com/dmlc/rabit

gorithm. However, the training set is composed of pairwise examples from each query group, which may cause very high communication cost when the examples are irrationally distributed across workers. To avoid this problem, we propose two methods to divide the training samples appropriately.

The first method is to use a multiway number partitioning algorithm [5] to divide the query ids into different workers. Then, samples with the same query id are assigned to the same worker where the id belongs to. As a result, the training samples are divided as evenly as possible, and the workload on workers is balanced. The detailed steps are listed as follows: 1) For each query id, scan the whole dataset to count the number of samples with it; 2) Each worker calculates the partition according to the multiway number partitioning algorithm [5]; 3) Each worker loads the training samples with query ids which belongs to the worker.

The second method is an approximate method, in which we need to store the samples of the same query id continuously in the file system. Then each worker loads evenly divided data as usual. If samples with the same query id are assigned into two workers, they will be treated as samples with two different query ids. Even though the approximate method could lose some pairwise samples, it empirically works well for most of our tasks.

4. PERFORMANCE

In this section, we report the performance of our PSMART on four industrial datasets on problems of CTR (click through rate) and CVR (click value rate), which are summarized in Table 1. We also successfully tested even bigger datasets with hundreds of billions of samples, which XGBoost cannot handle at all.

Table 1: Dataset description and statistics.

Dataset	Description	#Feat.	#Sample(M)
Rec CTR	Items recommendation	78	84
Search CTR	Search results ranking	592	1,000
Ads CVR1	Search ads ranking	698	2,067
Ads CVR2	Search ads ranking	698	10,245

We adopt the well-known XGBoost as our baseline. Since both methods can achieve basically the same AUC (e.g. 0.7368 for Ads CVR2 set) at state-of-the-art level without statistically significant differences, they are mainly compared by memory consumption and computational time which are two key factors in the production environment. To make a fair comparison, all algorithms adopt the same experimental setup. Specifically: tree number=500, max depth=6, tree method=approximate split-finding, min child weight=100, data and feature sample rate=1.0, bin number=128. All the experiments are conducted on the same production cluster with two twelve-core Intel Xeon CPU E5-2430 (2.2GHz) and 96GB of memory (all the data are loaded into memory).

Memory consumption: The peak total memory used are summarized in Table 2, which shows PSMART consumes slightly less memory than XGBoost. Moreover, XGBoost are tried on Ads CVR2 more than ten times and never succeed. The reason is that, when the dataset is very large, many computation nodes are needed so that the probability of failures of some nodes is very high. But, Rabit only offers limited fault tolerance so that when the tracker node fails it can not recover, but our PSMART system supports robust failover mechanism so that it can recover easily.

Computational time: Figure 1 shows the running time of all the algorithms on all the datasets. Here, we observe

Table 2: Peak total memory use on training sets.

Alg. \ Data	Rec CTR	Search CTR	Ads CVR1	Ads CVR2
PSMART	162G	1,775G	3,711G	22,157G
XGBoost	176G	1,810G	3,912G	N/A

that PSMART is significantly more efficient than XGBoost. This is because that the PS system enjoys sparse communication and traffic control optimizations, which make PS more efficient than Rabit. Thus the waiting time of PSMART is extremely reduced and the CPU resources are fully utilized.

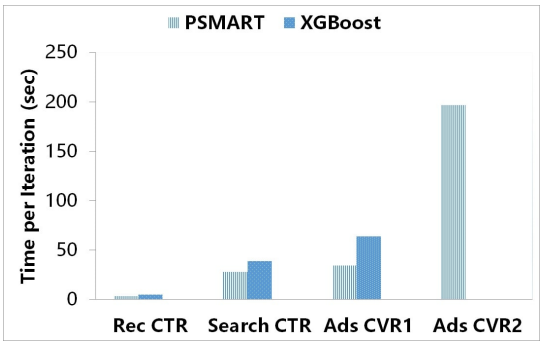

Figure 1: Runtime comparison on four training sets.

Finally, we would like to emphasize that when the scale of dataset becomes greater, more computation nodes are needed so that the probability of failures of some nodes is highly increased. However, if the tracker node failed for XGBoost, the whole job can not be recovered quickly, which makes it unsuitable to extremely big training data. Conversely, our PSMART system dumps the intermediate model and local variables into distributed memory, thus the training can be recovered from the nearest iteration easily. Due to these difference, our PSMART enjoys higher scalability and robustness than XGBoost, so that it is more suitable for industrial production applications.

5. CONCLUSION

We introduce a PS based distributed learning system, PSMART, that reliably scales MART to support hundreds of billions of samples and thousands of features in production. Experiments in Alibaba's production environment validates the advantages of PSMART in memory and time consumption against the state-of-the-art XGBoost.

6. REFERENCES

[1] C. J. Burges. From ranknet to lambdarank to lambdamart: An overview. *Learning*, 11:23–581, 2010.

[2] T. Chen and C. Guestrin. Xgboost: A scalable tree boosting system. In *SIGKDD, 2016*.

[3] D. Cossock and T. Zhang. Statistical analysis of bayes optimal subset ranking. *Trans. Infor. Theory, 2008*.

[4] P. Jiang and et al. Life-stage prediction for product recommendation in e-commerce. In *KDD*, 2015.

[5] R. E. Korf. Multi-way number partitioning. In *IJCAI 2009*.

[6] M. Li, D. G. Andersen, A. J. Smola, and K. Yu. Communication efficient distributed machine learning with the parameter server. In *NIPS*, 2014.

[7] Microsoft. https://github.com/microsoft/lightgbm.

[8] E. P. Xing and et al. Petuum: A new platform for distributed machine learning on big data. In *KDD*, 2015.