Challenges and Opportunities of Building Fast GBDT Systems

Zeyi Wen[1] , **Qinbin Li**[2] , **Bingsheng He**[2] and **Bin Cui**[3]
[1]The University of Western Australia
[2]National University of Singapore
[3]Peking University
zeyi.wen@uwa.edu.au, {qinbin, hebs}@comp.nus.edu.sg, bin.cui@pku.edu.cn

Abstract

In the last few years, Gradient Boosting Decision Trees (GBDTs) have been widely used in various applications such as online advertising and spam filtering. However, GBDT training is often a key performance bottleneck for such data science pipelines, especially for training a large number of deep trees on large data sets. Thus, many parallel and distributed GBDT systems have been researched and developed to accelerate the training process. In this survey paper, we review the recent GBDT systems with respect to accelerations with emerging hardware as well as cluster computing, and compare the advantages and disadvantages of the existing implementations. Finally, we present the research opportunities and challenges in designing fast next generation GBDT systems.

1 Introduction

The successful development of machine learning recently is not only due to better algorithms with high model quality, but also efficient algorithms and systems which use the high-performance hardware (e.g., GPUs and FPGAs) and a cluster of servers. Nowadays, many companies (e.g., Amazon, Google, Microsoft and Oracle) are providing GPU clouds as an integral component in cloud computing infrastructures. More and more researchers are exploring emerging hardware and/or clouds for machine learning algorithms such as deep learning [Abadi *et al.*, 2016; Zhang *et al.*, 2020] and SVMs [Wen *et al.*, 2018b].

Gradient Boosting Decision Trees (GBDTs) have been accelerated by multi-core CPUs, GPUs and clusters. GBDTs have become one of the most popular machine learning models recently, due to its excellent capability of handling tabular data. GBDTs have been widely used in various applications such as online advertising and spam filtering. In this survey, we focus on the challenges and opportunities on building fast GBDT systems with parallel processing on emerging hardware such as multi-core CPUs and GPUs as well as distributed computing in the computing cluster.

There have been a number of libraries for GBDTs (e.g., XGBoost [Chen and Guestrin, 2016], LightGBM [Ke and et al., 2017], CatBoost [Dorogush *et al.*, 2018a] and ThunderGBM [Wen *et al.*, 2020]). While those libraries offer similar functionalities for GBDT training and predictions, their algorithmic design and system implementation are quite different. For example, while many of them can take advantage of GPU computation power, they have very different choices in which part of model training and prediction to run on the GPU and the strategies of growing the trees. Moreover, whether GPU implementations are faster than multi-core CPU implementation is problem dependant [Ke and et al., 2017; Wen *et al.*, 2020]. GPU implementations tend to be more advantageous in medium size problems which can fit in the GPU memory, while CPU implementations tend to be more scalable in very large problems.

In order to understand the current design of different GBDT systems and to identify the future research opportunities for next-generation GBDT systems, we conduct a survey on existing GBDT systems running on parallel and distributed environments. This survey is timely in reviewing the effort in existing GBDT systems, and in architecting future systems. Specifically, we identify a few research challenges raised from irregular memory accesses in training, the support for online GBDT learning and privacy leakage in the trees, as well as handling workload balancing among threads. With challenge comes opportunity, we have seen the opportunities coming from fast networks and hardware evolution. For example, no mature GBDT library exploits FPGAs which are becoming increasingly powerful for machine learning. Moreover, advanced network equipment, such as Remote Direct Memory Access (RDMA), can potentially help resolve the network bottleneck in GBDT training.

2 Background

GBMs were introduced by Friedman (2001), and have shown great potential in many real world applications. The GBDT is an ensemble model which trains a sequence of decision trees, where the later tree aims to correct the errors of the previous trees. GBDTs are widely used in various applications especially those with real-time requirements, such as advertising systems, fraud detection, spam filtering, and image labeling [Chen and Guestrin, 2016].

Each decision tree is built from the depth of 0 to the maximum depth. The greedy based node splitting algorithm is used to recursively grow the trees until the termination condition is met (e.g., hitting the maximum depth or further splitting having no error reduction). The approaches of finding the split point for each node include (i) the exact approach, and (ii) the approximate approach. In the exact approach, GBDTs traverse all the feature values to find the split that maximizes the gain. The cost of finding the exact split points is prohibitively high for data sets with high dimensions. In the approximate approach, GBDTs only try some candidate points generated by the techniques such as quantile sketches [Chen and Guestrin, 2016] or histograms [Ke and et al., 2017].

3 Parallel GBDT Training

Developing efficient parallel algorithms for GBDT training is challenging, due to the large number of irregular memory accesses. The key challenges of developing efficient parallel GBDTs include (i) minimizing communication cost among computing units, (ii) making efficient use of the resources of the computing units, and (iii) adapting to the problem properties (e.g., data sparsity). Here, we review GBDT parallel training techniques and systems that run on multi-core CPUs and GPUs in a single machine.

3.1 Node Split

A traditional way of finding the best split point for a node is through enumerating all the possible split points. This way of finding the best split point is expensive for problems with a large number of feature values, because essentially every feature value needs to be evaluated when finding the best split point. One common solution in the current parallel GBDT systems is using histogram based approximation instead of enumerating all the possible split points, which can be viewed as considering a number of representative split points using histogram based approximation. This approximation is a trade-off between efficiency and accuracy [Chen and Guestrin, 2016]. However, this solution is ill-suited to high dimensional and sparse problems, since each dimension needs to associate with a histogram. In a parallel algorithm, the total number of histograms needed to be maintained equals to: the number of threads multiplied by the number of features. Hence, the parallelism granularity, finding best split by enumeration or by histograms, etc., also need to adapt to this difference in order to achieve high computation resource utilization. Existing systems are either specifically optimized for a single machine or performing badly in high dimensional applications.

3.2 Parallelism Granularity

The key granularities for the parallelism include three levels: (i) node level parallelism, (ii) feature level parallelism and (iii) value level parallelism. In the node level parallelism, one thread is dedicated to finding the best split of a node, and hence it evaluates the gains of all the values for all the features in a node. In the feature level parallelism, one thread is dedicated to finding the best value for only one feature of a node. Hence, one thread evaluates the gains of all the values for one feature of a node. In the value level parallelism, one thread evaluates the gain of one value of a feature in a node. The node level and feature level parallelism are often adapted by multi-core CPU based training algorithms. The mainstream GPU implementations use the value level parallelism which is the finest granularity of parallelism in finding the best split.

3.3 Training Data Partitioning

When the training problem is tackled by multiple GPUs or multiple CPUs, the training data set needs to be partitioned. There are three ways of partitioning the data: instance based partitioning, feature based partitioning, and hybrid instance and feature based partitioning. Instance based partitioning assigns a number of training instances to a machine, and the rest to the other machines. The benefit of this way of partitioning is that each machine has the whole information of the instance stored in it. Hence, when computing the training error for an instance, the machine storing the instance can easily compute it. In comparison, feature based partitioning assigns all the values of a number of features to one machine. Thus, computing the training error is much more difficult, as a machine does not have all the information of an instance. However, the advantage of the feature based partitioning is that a machine can easily find the best split value of a feature, as all the feature values of the training data are stored together. The hybrid partitioning approach considers the training data set as a matrix, and each GPU/machine handles a block or tile [Vasiloudis *et al.*, 2019]. The mainstream GBDT systems user either instance based partitioning or feature based partitioning. The hybrid partitioning approach is not widely adopted, due to the implementation and maintenance complexity.

3.4 GBDT Training on CPUs

On the parallel GBDT training on CPUs, irregular tree structure leads to a number of challenges in designing an efficient GBDT library. First, due to the nature of tree structures, the memory access pattern of GBDT training is irregular. The training instances are divided into different nodes after each splitting. Thus, it is hard to guarantee continuous memory read and write operations when accessing the instances in a node. The irregular memory accesses can significantly degrade the efficiency of the training process. Second, sorting feature values during the GBDT training may be very costly. The feature values of every node usually need to be sorted in order to speed up the enumeration of possible split points. Since every feature in every node needs to be sorted, the number of sorting operations with small inputs is huge when handling high dimensional problems. Therefore, the total number of sorting operations equals to: the number of nodes multiplied by the number of features, which is huge for high dimensional problems. Lastly, it is hard to design a workload balanced parallel algorithm. The number of nodes in each depth may be different and the number of instances also varies in different nodes. The data parallel granularity changes as the tree grows.

To address those challenges, the existing libraries such as XGBoost, LightGBM and CatBoost can run on CPUs in parallel and adopt different designs on parallelism, node split and training data partitioning. XGBoost has the best scalability to the number of machines, LightGBM prunes unimportant instances and features during training, and CatBoost aims to train trees for faster prediction. Basically, XGBoost finds the best split points for multiple attributes of multiple nodes concurrently. In other words, the GBDT systems on CPUs use both attribute level and node level parallelism, which may result in more than enough threads to occupy the CPUs. LightGBM and CatBoost have similar parallelism principles as XGBoost. The key difference is that LightGBM eliminates some instances with small gradients and combines correlated attributes, and CatBoost trains the so-called oblivious trees where an identical split point is used in the whole level of a tree. Recent studies exploit both value-level parallel and feature-level parallelism at different stages of the training [Peng *et al.*, 2019], and exploit bit-level optimization for GBDTs [Devos *et al.*, 2019]. The authors showed that the proposed algorithms can outperform CPU-based XGBoost and LightGBM. Along with the mainstream GBDT libraries, there is much research on training decision trees on CPUs. For example, Panda et al. (2009) leveraged the MapReduce framework for learning decision trees. Tyree et al. (2011) designed a method to parallel boosted regression trees for webpage ranking. Si et al. (2017) introduced a variant of GBDT algorithm for high dimensional sparse output.

3.5 GBDT Training on GPUs

Due to the architectural differences between CPUs and GPUs, a lot of work has been done on optimizing parallelism, node split and training data partitioning of GBDT training in order to exploit the GPU power. Existing GBDT systems which can run on GPUs include XGBoost, LightGBM, CatBoost and ThunderGBM. Here, we present their key ideas of parallelism on GPUs. Similar to the CPU version, XGBoost on GPUs also uses attribute level and node level parallelism. For attribute level parallelism, a GPU thread block is for computing the best split point of an attribute. For node level parallelism, XGBoost uses node interleaving techniques on GPUs [Mitchell and Frank, 2017]. However, XGBoost requires reserving many copies of the gradient of each instance, where the number of copies equals to the number of nodes to split. Moreover, they use the dense data format for the training data set, which can easily track back the attribute of the best split point. Therefore, this way of parallelism on GPUs requires too much GPU memory and cannot handle large data sets. To tackle this problem, XGBoost applies the histogram based approximation to reduce the memory consumption. LightGBM and CatBoost also use finding the best split points approximately to avoid consuming too much memory while enumerating all the possible split points. ThunderGBM is the latest GBDT library on GPUs. It supports multiple GPUs and has been reported to produce comparable models while faster than the other libraries. ThunderGBM can handle high dimensional data efficiently, while the other libraries fail [Wen *et al.*, 2020].

There is also research on accelerating decision tree training using GPUs, although not specifically for GBDTs. For a random forest, Grahn et al. (2011) proposed to use a GPU thread to train each decision tree. Since the trees do not have a dependency on each other unlike GBDTs, they can be trained in parallel. Nasridinov et al. (2014) developed a GPU-based algorithm to compute the information gain of each split point of a node. Lo et al. (2014) proposed to split one node per iteration. The values of each attribute for all the instances are sorted in the node. However, GPU resources can be underutilized, since the level of parallelism is notably low. Strnad and Nerat (2016) took advantage of three levels of parallelism in the GPU computing: 1) evaluating multiple split points simultaneously, 2) finding the best split point for multiple attributes on a node in parallel, 3) finding the best attribute for multiple nodes concurrently. Their algorithm launches many kernels inside GPU kernels, which introduces significant overhead. Moreover, the attribute values are repeatedly sorted for every new node. Most of the above methods for training decision trees are implemented in the GPU version of the existing GBDT systems such as XGBoost. The H2O system [Cook, 2016] adapts the GPU implementation of XGBoost. In contrast, Wen et al. (2018a; 2019) proposed approaches which use data compression techniques to efficiently train GBDTs and to support larger data sets.

Existing studies use GPUs to optimize the decision tree prediction which is part of GBDT training when computing the gradients. Sharp used GPUs for accelerating the decision trees and forests prediction [Sharp, 2008]. The main idea is for each GPU thread to predict the target value of a single instance, which takes advantage of the massive thread parallelism on the GPU. Like Sharp's algorithm, Birkbeck et al. (2011) studied a GPU-based algorithm for the decision tree prediction. The decision tree is stored in GPU texture memory to improve efficiency.

3.6 System Comparison

Table 1 summarizes the system features of GBDT training algorithms on GPUs. Systems support different features in parallelism, handling sparsity data and node split.

Tabel 2 shows the performance comparison between existing GBDTs implementations downloaded on 4 Jan 2021. We conducted the experiments on a machine running Linux with two Xeon E5-2640v4 10 core CPUs, 256GB main memory and one Tesla Pascal P100 GPU of 12GB memory. The data sets are publicly available from the LIBSVM website. The prediction accuracies are similar between different libraries in all the experiments. LightGBM and CatBoost tend to have lower prediction accuracy compared with XGBoost and ThunderGBM. We have two major observations in terms of efficiency from the results. First, compared with running on CPUs, all the libraries can gain significant improvement in efficiency by running on GPUs. Second, LightGBM appears to usually work better than other existing libraries when running on CPUs, while ThunderGBM has a better performance when running on GPUs especially on high dimensional data and on large data sets. The GPU implementation of CatBoost is excellent, but we also observed that its GPU memory consumption is significantly higher than other implementations. Hence, CatBoost ran out of memory (denoted by "oom") on the *news20* data set which has about 60 thousand dimensions.

4 Distributed GBDT Training

The current GBDT systems such as XGBoost, LightGBM, Spark MLlib and DimBoost can run on a distributed environment. They use the instance based data partitioning, where

[1]https://github.com/Microsoft/LightGBM/

[2]https://catboost.ai/#benchmark

[3]https://github.com/Xtra-Computing/thundergbm

system name	sparsity aware	multi-gpu support	find exact split points	support text data	compress data	regular tree	best first	distributed gpus	categorical preprocessing
xgboost	✓	✓	✗	✗	✗	✓	✓	✓	✗
lightgbm	✓	✗	✗	✗	✗	✓	✓	✓	✓
catboost	✗	✓	✗	✓	✗	✗	✗	✗	✓
thundergbm	✓	✓	✓	✗	✓	✓	✗	✗	✗

Table 1: Comparison between existing GPU implementations of GBDTs. XGBoost on GPU is considered not supporting finding exact split points, as it either runs out of memory or produces extremely large RMSE.

data set			Training on CPUs only (sec)			Training accelerated with GPUs (sec)			
name	card.	dim.	XGBoost	LightGBM	CatBoost	XGBoost	LightGBM	CatBoost	ThunderGBM
abalone	4177	8	1.3	**0.1**	0.4	1.3	1.2	1.8	0.72
covetype	581K	54	3.1	4.3	3.7	**1.0**	11	1.5	1.7
cpusmall	8192	12	1.4	**0.1**	0.5	1.4	1.2	1.8	0.7
insurance	13M	35	23	27	47	9	25	10	**6**
higgs	11M	28	45	48	39	10	40	8	**6.6**
news20	16K	62K	287	**15.4**	oom	109	16.5	oom	16.5
susy	5M	18	17	35	22	1.9	28	3.6	**3.1**

Table 2: Efficiency comparison between existing GBDTs implementations (downloaded on 4 Jan 2021)

each machine stores a number of training instances. In the following, we first discuss key system issues including load balancing, communication of different data partitioning, and heterogeneous clusters, followed by system comparison.

4.1 Load Balancing

Load balancing in terms of the number of instances or features. The instance based data partitioning divides the training data in the following manner. Suppose the cluster has two machines, the instance based data partitioning assigns $\frac{n}{2}$ instances to one machine and the remaining training instances to the second machine. The feature based data partitioning assigns $\frac{d}{2}$ features to one machine and the remaining features to the second machine.

Load balancing in terms of the number of feature values. An alternative way of balancing the workload is through the number of feature values. For instance based data partitioning, a number (denoted by n_1) of training instances are assigned to one machine and the remaining $(n - n_1)$ training instances are assigned to the second machine, such that the total number of feature values in each of the machines is the same or similar. For feature based data partitioning, a number (denoted by d_1) of features are assigned to a machine and the remaining $(d - d_1)$ features are assigned to another machine, such that the two machines have the similar number of feature values.

4.2 Network Communication

Network communication is mainly from two aspects: constructing the histogram for a feature and finding the best feature among all the features. We present them in the two ways of partitioning data.

Instance based data partitioning. The values of a feature are distributed in different machines. Hence, finding the best split value for a feature requires network communication between machines. The instance based data partitioning cannot support finding the best split point by enumeration, since a machine does not have all the available information about the feature to perform the enumeration. Hence, we only discuss how to find the best split using histogram based approximation [Jiang *et al.*, 2018]. The process of constructing the histogram for a feature works as follows. First, each machine constructs a local histogram for a feature using the instances stored locally. Then, the local histograms for a feature from each machine are aggregated by a centralized server in order to construct the histogram of the feature. The best split point of the feature can be computed by the server. Finally, the best split points of each feature are compared in the centralized server, and the best feature is selected.

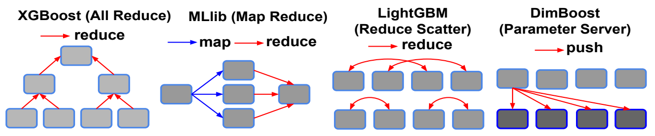

Figure 1: Different ways of communication in GBDT training

Feature based data partitioning. The values of a feature are fully stored in one machine. Hence, finding the best split value for a feature can be handled locally. Then, obtaining the best feature is simply collecting the best split value of each feature and to select the feature with the maximum gain. One question that may be raised is that one machine may not be able to store the values of a feature. This question is valid, but in reality, one machine has sufficient memory to store all the values of a feature. Let us take a more concrete example. Suppose a machine has only 10GB of memory for GBDT training, and each feature value requires 4 bytes. Then, a machine can store about 2.5 billion feature values, which indicates that the training data set has more than 2.5 billion instances.

Figure 1 shows four typical ways of communications used in the four GBDT training systems. XGBoost, DimBoost [Jiang

GBDT distributed implementation	Hadoop support	Spark support	SGE support	instance based data partition	feature based data partition	communication design
XGBoost	✓	✓	✓	✓	✗	centralized
LightGBM	✓	✓	✗	✓	✓	decentralized
MLlib [Meng and et al., 2016]	✓	✓	✗	✓	✗	centralized
DimBoost [Jiang *et al.*, 2018]	✓	✗	✗	✓	✗	centralized

Table 3: Comparison between existing implementations of GBDTs on some major distributed environments.

GBDT training algorithm	communication cost proportional to		
	# of ins.	# of dim.	user control parameter
XGBoost	✗	✓	✗
LightGBM	✗	✓	✗
CatBoost	✗	✓	✗
ThunderGBM	✓	✗	✗
MLlib	✗	✓	✗
DimBoost	✗	✓	✗
Vero	✓	✗	✗
PV-Tree	✗	✗	✓

Table 4: Main factor of communication complexity

et al., 2018] and Spark MLlib [Meng and et al., 2016] uses the centralized design, where the global information needs to be sent to one or multiple servers. In comparison, LightGBM uses a decentralized method for data communication. ThunderGBM also uses the centralized design for a machine with multiple GPUs, but does not support distributed computing.

Communication Complexity Analysis

Meng et al. (2016) proposed a communication efficient parallel decision tree training algorithm called "PV-Tree" which can be used in GBDT training as well. Here, we compare the communication complexity of different algorithms for training GBDTs. The most common ways of partitioning the training instances are: feature based partitioning and instance based partitioning [Fu *et al.*, 2019]. Feature based partitioning does not require the whole histogram to be sent to other machines. Instead, only the local best split is sent out in order to identify the global best split. The main communication cost is broadcasting the instance placement after a node split, which requires sending out all the IDs of the training instances. Therefore, the communication on the tree construction is $\mathcal{O}(N \times W \times L)$, where N is the number of training instances, W is the number of machines and L is the number of layers of the trees (a.k.a. tree depth). In comparison, the main communication cost of the instance based partitioning is sending out the histograms to other machines. We denote the size of the histograms of a tree node by $Size_{hist}$. Then the communication cost is $\mathcal{O}(Size_{hist} \times W \times 2^{L-1})$. The size of the histograms of a tree node depends on the number of dimensions D, the number of candidate splits per feature q, and the number of classes of the learning problem C.

4.3 System Comparison

Tabel 3 shows the features of some existing libraries which can run on a distributed environment. We investigate whether the libraries can run on some major distributed environments such as Apache Hadoop, Apache Spark and SGE (Sun Grid Engine). A few observations can be made. First, XGBoost has a complete development on the distributed systems. It supports many different distributed environments. Second, all the four libraries adopt the instance based data partition, where the instances are divided horizontally to different machines. Third, apart from LightGBM, the other libraries implement data communication in a centralized way. Although the instance based data partition and the centralized design seem to be the mainstreams in the implementations of GBDTs, there is no final conclusion as to which technique is better.

For performance comparison, the previous study [Jiang *et al.*, 2018] has presented some results. In the study, they compared DimBoost with XGBoost, LightGBM, and MLlib. Two clusters and three data sets are used in the experiments. From the results, MLlib is much slower than the other libraries, and DimBoost outperforms other libraries. Furthermore, DimBoost achieves the fastest convergence rate, followed by LightGBM. XGBoost converges slowly for high dimensional data.

5 Challenges and Opportunities

We present the challenges and opportunities GBDT systems.

5.1 Challenges

Benchmark for GBDT. Although there have been quite some efforts in developing libraries for GBDTs, a fair and robust benchmark is still a quite challenging open problem. Dorogush et al. (2018b) pointed out the main problems of current approaches of benchmarking GBDT libraries. We believe that a fair benchmark will be important for further development, evaluations and optimizations of GBDT libraries.

Privacy concerns. The GBDT model is relatively transparent [Li *et al.*, 2020b]. When the model is deployed, users can extract information relatively easily from the decision trees. Training data is very precious. Even worse, data are dispersed over different organizations in reality. For example, people tend to go to nearby hospitals, and the patient records in different hospitals can become "island". Hospitals may benefit from each other if they can collaborate to train a model with all their data as input. However, it is a non-trivial task for organizations to combine their data. For example, to protect user data privacy, data regulations and policies such as General Data Protection Regulation (GDPR) have posed restrictions on data movement among different parties. Recently, much research efforts have been devoted to developing new learning algorithms in the *federated learning* setting [Li *et al.*, 2020a]. It is still an open problem of how to develop GBDT systems under the context of federated learning.

Automated feature engineering and parameter tuning. Features and hyper-parameters of GBDTs such as the number of trees and tree depths often have a significant impact on the model efficacy. Feature augmentation has shown improvement for the predictive accuracy of GBDTs [Tannor and Rokach, 2019]. Therefore, features and hyper-parameters need to be carefully chosen. For hyper-parameter tuning, grid search technique is rather costly. One may develop/use Bayesian optimization to automate hyper-parameter tuning. Moreover, GBDTs rely on (handcrafted) feature engineering. To make machine learning more accessible to wider communities, automatic feature engineering is an emerging area. Featuretools [Kanter and Veeramachaneni, 2015] is an initiative of the area, but more techniques and research are needed in the field. GBDT systems with automatic feature engineering and parameter tend to be more useful for real-world applications.

Incremental learning and online learning. A total training may not be practical in some scenarios. One example is that the data is streaming data, where the training data is continuously generated. For such scenarios, incremental learning and online learning can be more effective. A recent proposal along this direction is that a new tree is built to correct the errors of the newly arrived data [He and et al., 2013]. However, this approach may result in a large number of trees, as the training proceeds. Ke et al. (2019) proposed techniques to extract knowledge from GBDTs to learn a Neural Network, in order to make use of the property of Neural Networks for online learning. However, retaining the tree structure may be crucial for applications with real-time inference constraints. The existing GBDT systems do not support online or incremental learning. Hence, more research needs to be carried out to improve the current GBDT systems for online learning.

Approximation. When the number of training instances is too large, the training process is either too slow or consumes too much memory. The common approximation technique is using histograms to approximate the data. Also, only a number of representative split points are considered. Thus, the computation cost is reduced. However, the memory consumption issue is not totally addressed for high dimensional data, since each dimension requires building a histogram. The memory consumption for the histograms is huge and may exceed the memory limits of existing GBDT systems like CatBoost. Thus, more approximation mechanisms can be developed with the minimum loss in the precision of the trained model.

5.2 Opportunities

Fast networks. Remote Direct Memory Access (RDMA) offers high performance networking fabrics in clusters and data centers. RDMA achieves very low round-trip latency (several μs), high throughput, and very low (almost zero) CPU overhead. For training large-scale GBDTs, RDMA can be leveraged to reduce the network bottleneck.

Heterogeneous Cluster with GPUs. Nowadays, many servers in a cluster are equipped with both CPUs and GPUs. Existing systems such as XGBoost, LightGBM and CatBoost can make use of both CPUs and GPUs of the servers. The current strategies of those systems to use CPUs and GPUs are standard. More specifically, the current systems use CPUs for data preparation and/or node splitting and use GPUs for finding the best split points of a node, as finding the best split is the most computationally expensive operation. More research on unleashing the capacity of CPUs and GPUs will be compelling in training GBDTs collaboratively.

FPGA on the Cloud. FPGAs have been widely adopted in industries to accelerate applications, due to their high energy efficiency and performance. Cloud FPGA providers such as Amazon, Baidu, and Tencent have also offered virtual machines with FPGAs. Some preliminary study [Tanaka and et al., 2018] used FPGA to accelerate GBDT, which has demonstrated promising results. More work can be done along this direction to further power the GBDT systems.

The emergence of artificial intelligence accelerators. Besides GPUs and FPGAs, AI specialized hardware accelerators are emerging in recent years, such as Tensor Cores and Google TPU[4]. These hardware accelerators have been used to accelerate machine learning applications, especially for deep learning in the cloud. The experiences and knowledge created from those specialized architectures lead to new thoughts in further improving the energy efficiency of GBDT systems. It is worth to explore whether specialized hardware is cost-effective for GBDTs in terms of training and prediction efficiency.

6 Conclusions

Fast GBDT systems is a hot research topic that has been gaining considerable popularity in recent years. A number of systems have been developed to improve the efficiency of GBDT training on a single machine with multi-core CPUs and GPUs, as well as on a cluster of servers. *To the best of our knowledge, there is no systematic survey among those GBDT training systems.* In this work, we have presented a timely survey study to summarize the existing effort and point out the future direction. We have compared the differences and common designs among those GBDT systems. Moreover, we present the comparison on system features as well as training performance. We believe that hardware-software codesign is an important aspect for fast GBDT systems. More importantly, we have pointed out a number of challenges and opportunities, which call for the effort from machine learning, computer architecture and computer system communities.

Acknowledgements

This research is supported by Oracle for Research and the National Research Foundation, Singapore under its AI Singapore Programme (AISG Award No: AISG2-RP-2020-018). Any opinions, findings and conclusions or recommendations expressed in this material are those of the authors and do not reflect the views of National Research Foundation, Singapore. Bin Cui's work is supported by NSFC No.61832001, PKU-Baidu Fund 2019BD006, and Beijing Academy of Artificial Intelligence (BAAI). We extend our great gratitude to our team members of ThunderGBM and DimBoost. We also thank the insightful comments and suggestions from Dr. Guolin Ke.

[4]http://cloud.google.com/tpu/, Jan. 2019.

References

[Abadi *et al.*, 2016] Martín Abadi, Paul Barham, and et al. TensorFlow: A system for large-scale machine learning. In *OSDI*, 2016.

[Birkbeck and et al., 2011] Neil Birkbeck and et al. Fast boosting trees for classification, pose detection, and boundary detection on a GPU. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR) Workshops*. IEEE, 2011.

[Chen and Guestrin, 2016] Tianqi Chen and Carlos Guestrin. XGBoost: A scalable tree boosting system. In *ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD)*, 2016.

[Cook, 2016] Darren Cook. *Practical machine learning with H2O: powerful, scalable techniques for deep learning and AI.* " O'Reilly Media, Inc.", 2016.

[Devos *et al.*, 2019] Laurens Devos, Wannes Meert, and Jesse Davis. Fast Gradient Boosting Decision Trees with bit-level data structures. In *Joint European Conference on Machine Learning and Knowledge Discovery in Databases (ECML/PKDD)*. Springer, 2019.

[Dorogush *et al.*, 2018a] Anna Veronika Dorogush, Vasily Ershov, and Andrey Gulin. CatBoost: gradient boosting with categorical features support. *arXiv preprint arXiv:1810.11363*, 2018.

[Dorogush *et al.*, 2018b] Anna Veronika Dorogush, Vasily Ershov, and Dmitry Kruchinin. Why every GBDT speed benchmark is wrong. *CoRR*, abs/1810.10380, 2018.

[Friedman, 2001] Jerome H Friedman. Greedy function approximation: a gradient boosting machine. *Annals of statistics*, pages 1189–1232, 2001.

[Fu *et al.*, 2019] Fangeheng Fu, Jiawei Jiang, Yingxia Shao, and Bin Cui. An experimental evaluation of large scale GBDT systems. *Proceedings of the VLDB Endowment (PVLDB)*, 12(11):1357–1370, 2019.

[Grahn and et al., 2011] Håkan Grahn and et al. CudaRF: a CUDA-based implementation of random forests. In *AICCSA*, 2011.

[He and et al., 2013] Miao He and et al. Robust online dynamic security assessment using adaptive ensemble decision-tree learning. *IEEE Transactions on Power systems*, 2013.

[Jiang *et al.*, 2018] Jiawei Jiang, Bin Cui, Ce Zhang, and Fangcheng Fu. DimBoost: Boosting Gradient Boosting Decision Tree to higher dimensions. In *SIGMOD*, 2018.

[Kanter and Veeramachaneni, 2015] James Max Kanter and Kalyan Veeramachaneni. Deep feature synthesis: Towards automating data science endeavors. In *DSAA*, pages 1–10. IEEE, 2015.

[Ke and et al., 2017] Guolin Ke and et al. LightGBM: A highly efficient gradient boosting decision tree. In *Advances in Neural Information Processing Systems (NeurIPS)*, pages 3149–3157, 2017.

[Ke *et al.*, 2019] Guolin Ke, Zhenhui Xu, Jia Zhang, Jiang Bian, and Tie-Yan Liu. Deepgbm: A deep learning framework distilled by GBDT for online prediction tasks. In *ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD)*, pages 384–394. ACM, 2019.

[Li *et al.*, 2020a] Qinbin Li, Zeyi Wen, and Bingsheng He. Practical federated GBDTs. In *AAAI Conference on Artificial Intelligence (AAAI)*, 2020.

[Li *et al.*, 2020b] Qinbin Li, Zhaomin Wu, Zeyi Wen, and Bingsheng He. Privacy-preserving GBDTs. In *AAAI Conference on Artificial Intelligence (AAAI)*, 2020.

[Lo and et al., 2014] Win-Tsung Lo and et al. CUDT: a CUDA based decision tree algorithm. *The Scientific World Journal*, 2014, 2014.

[Meng and et al., 2016] Xiangrui Meng and et al. Mllib: Machine learning in Apache Spark. *Journal of Machine Learning Research (JMLR)*, 17(1):1235–1241, 2016.

[Meng *et al.*, 2016] Qi Meng, Guolin Ke, Taifeng Wang, Wei Chen, Qiwei Ye, Zhi-Ming Ma, and Tie-Yan Liu. A communication-efficient parallel algorithm for decision tree. In *Advances in Neural Information Processing Systems (NeurIPS)*, pages 1279–1287, 2016.

[Mitchell and Frank, 2017] Rory Mitchell and Eibe Frank. Accelerating the XGBoost algorithm using GPU computing. *PeerJ Preprints*, 5:e2911v1, 2017.

[Nasridinov *et al.*, 2014] Aziz Nasridinov, Yangsun Lee, and Young-Ho Park. Decision tree construction on GPU: ubiquitous parallel computing approach. *Computing*, 96(5):403–413, 2014.

[Panda and et al., 2009] Biswanath Panda and et al. Planet: massively parallel learning of tree ensembles with MapReduce. *Proceedings of the VLDB Endowment (PVLDB)*, 2009.

[Peng *et al.*, 2019] Bo Peng, Langshi Chen, Jiayu Li, Miao Jiang, Selahattin Akkas, Egor Smirnov, Ruslan Israfilov, Sergey Khekhnev, Andrey Nikolaev, and Judy Qiu. HarpGBDT: Optimizing GBDTs for parallel efficiency. In *CLUSTER*, pages 1–11. IEEE, 2019.

[Sharp, 2008] Toby Sharp. Implementing decision trees and forests on a GPU. In *European Conference on Computer Vision (ECCV)*, pages 595–608, 2008.

[Si and et al., 2017] Si Si and et al. Gradient Boosted Decision Trees for high dimensional sparse output. In *International Conference on Machine Learning (ICML)*, 2017.

[Strnad and Nerat, 2016] Damjan Strnad and Andrej Nerat. Parallel construction of classification trees on a GPU. *Concurrency and Computation: Practice and Experience*, 28(5):1417–1436, 2016.

[Tanaka and et al., 2018] Takuya Tanaka and et al. Efficient logic architecture in training Gradient Boosting Decision Tree for high-performance and edge computing. *arXiv preprint arXiv:1812.08295*, 2018.

[Tannor and Rokach, 2019] Philip Tannor and Lior Rokach. Augboost: gradient boosting enhanced with step-wise feature augmentation. In *International Joint Conference on Artificial Intelligence (IJCAI)*, pages 3555–3561, 2019.

[Tyree and et al., 2011] Stephen Tyree and et al. Parallel boosted regression trees for search ranking. In *International Conference on World Wide Web (WWW)*, 2011.

[Vasiloudis *et al.*, 2019] Theodore Vasiloudis, Hyunsu Cho, and Henrik Boström. Block-distributed gradient boosted trees. In *International ACM SIGIR Conference on Research and Development in Information Retrieval (SIGIR)*, pages 1025–1028. ACM, 2019.

[Wen *et al.*, 2018a] Zeyi Wen, Bingsheng He, Ramamohanarao Kotagiri, Shengliang Lu, and Jiashuai Shi. Efficient Gradient Boosted Decision Tree training on GPUs. In *International Parallel and Distributed Processing Symposium (IPDPS)*, pages 234–243. IEEE, 2018.

[Wen *et al.*, 2018b] Zeyi Wen, Jiashuai Shi, Qinbin Li, Bingsheng He, and Jian Chen. ThunderSVM: A fast SVM library on GPUs and CPUs. *Journal of Machine Learning Research (JMLR)*, 19:1–5, 2018.

[Wen *et al.*, 2019] Zeyi Wen, Jiashuai Shi, Bingsheng He, Jian Chen, Kotagiri Ramamohanarao, and Qinbin Li. Exploiting GPUs for efficient GBDT training. *IEEE Transactions on Parallel and Distributed Systems (TPDS)*, 30(12):2706–2717, 2019.

[Wen *et al.*, 2020] Zeyi Wen, Hanfeng Liu, Jiashuai Shi, Qinbin Li, Bingsheng He, and Jian Chen. ThunderGBM: Fast GBDTs and random forests on GPUs. *Journal of Machine Learning Research (JMLR)*, 21(108):1–5, 2020.

[Zhang *et al.*, 2020] Chenyang Zhang, Feng Zhang, Xiaoguang Guo, Bingsheng He, Xiao Zhang, and Xiaoyong Du. imlbench: A machine learning benchmark suite for CPU-GPU integrated architectures. *IEEE Transactions on Parallel and Distributed Systems*, 2020.