Asynch-SGBDT: Train Stochastic Gradient Boosting Decision Trees in an Asynchronous Parallel Manner

Abstract—Boosting Decision Tree (GBDT) is a costly machine learning model. Current parallel GBDT algorithms generally follow a synchronous parallel design: Fork-join parallel manner, like MapReduce. Fork-join parallel manner needs considerable time. In this paper, we try to build a stochastic optimization problem by sampling, which shares the same output with original GBDT training problem and use asynchronous parallel SGD manner to train Gradient step GBDT. We name our algorithm as asynch-SGBDT. Our theoretical and experimental results indicate that compared with the serial GBDT training process, when the datasets' high sample diversity is high and using Gradient step training GBDT, asynch-SGBDT does not slow down convergence speed on the epoch, and the sample diversity of high-dimensional sparse datasets is usually high. We conduct experiments on a 32-node cluster using three different datasets (KDD2010, KDD2012, and real-sim datasets). The results show that with LightGBM using a single worker as the baseline, LightGBM (the state-of-the-art synchronous parallel algorithm) on 32 workers achieves 5x-7x speedup, while our asynch-SGBDT on 32 workers increases the speedup to 11x-15x.

Index Terms—GBDT, asynchronous parallel, SGD, parameter server

1 Introduction

Gradient Boosting Decision Tree (GBDT) is an effective nonlinear machine learning model, and it is widely used in different areas like CTR prediction [1], search engines [2] and other applications [3] [4]. However, training a GBDT model is a costly process. In essence, GBDT is the optimization process, and only Newton step method or Gradient step method, which are sequential algorithms, can train the GBDT model [5] [6] [7].

In Gradient/Newton Step, building a decision tree is the most expensive process. Building a decision tree is a memory-bound problem. To build the decision tree fast, different researchers propose high performance building tree methods, like histogram-based algorithm [8] [9] [10] [11], voting parallel methods [9]. Current methods in gaining computation efficiency rely heavily on synchronous parallel computing technology, i.e., fork-join parallel implementation.

However, the requirement for synchronization creates a bottleneck for further improvement in computation speed in GBDT application on current commodity clusters and data center environments. In real production environment, i.e.,commodity clusters and data center environments, the cluster's nodes are often various. Their data centers or production environment clusters consist of different kinds of nodes for different reasons. For example, in the companies like startup, their data centers are built in step-by-step manner at different times, which leads to that their clusters are more complex. In these heterogeneous clusters, synchronous operation is the bottleneck for the fast nodes have to be blocked before the slow nodes finish their work.

Asynchronous parallel methods are one of the important parallel methods to overcome the shortage of synchronous. In asynchronous parallel stochastic gradient descent, i.e., the SGD algorithm on Parameter Server, the current model adds the older model's gradient. The delay between the current and older models is a random number, which is bounded by a constant number. In Parameter Server, different workers are blind to each other to reduce the cost of synchronization. Asynchronous parallel SGD achieves a satisfying high performance.

Inspired by the success of asynchronous parallel stochastic gradient descent on the Parameter Server framework, we ponder the questions: Is synchronization necessary for GBDT training, and if a GBDT can be trained in an asynchronous parallel manner? Are asynchronous parallel methods effective in GBDT training process?

Unlike the existing works, which focus on parallelizing the tree building process, we try to put parallel technologies on training methods to replace Newton step and Gradient step: we propose a asynchronous parallel Gradient step that relaxes synchronous requirement. Our asynchronous parallel stochastic GBDT (asynch-SGBDT) method works on the parameter server framework: The server receives trees from workers, and different workers build different trees in an asynchronous parallel manner.

We develop theoretical analyses via connecting asynch-SGBDT training process with stochastic optimization problem: we adopt function space optimization into high-dimensional parameter space optimization following Sun et al.'s work [7], and we introduce a random variable into the objective function through sampling operation. Thus, asynchronous parallel stochastic gradient descent method can be used to train GBDT.

Theoretical analyses show that high diversity samples of the datasets, small sampling rate, small step length and a large number of leaves for the GBDT settings can lead to high scalability for asynch-SGBDT. We call such conditions as asynch-SGBDT requirements.

From the aspect of validity, our analyses and experiments suggest that compared with the Gradient step GBDT training process, when the datasets are high-dimensional sparse datasets, asynch-SGBDT does not slow down convergence speed on the epoch. The widely used datasets in the big data industry, which are high-dimensional and sparse, are likely to meet the asynch-SGBDT

requirements.

From the aspect of parallel efficiency, compared with LightGBM with one worker, our asynch-SGDBT experiments code, which is based on LightGBM and ps-lite, reaches 11x-15x speedup when it uses 32 workers on real-sim, KDD2010 and KDD2012 dataset, compared with LightGBM with one worker. As the benchmark, LightGBM, a state-of-art parallel GBDT framework, only achieves 5x-7x speedup when it uses 32 machines, compared with LightGBM with one worker.

Our main contributions are the following: (1) We propose asynch-SGBDT, a Gradient step based asynchronous parallel method that can be used to train the GBDT model more efficiently. (2) We provide theoretical justification for asynch-SGBDT through an analysis of the relationships among convergence speed, number of workers and algorithm scalability. (3) Our approach achieves the same accuracy and higher efficiency than traditional Gradient step GBDT training method and its synchronous parallel implement on our tested datasets which are high dimensional and sparse.

2 Related Works and the Problem of Current Algorithm

The Gradient Boosted Decision Tree (GBDT) is developed by Friedman [5]. GBDT is an important Ensemble Learning model. Other famous Ensemble Learning machine learning models include Adaboost, Random Forest [12]. Boosting theory is developed slowly but solidly: Although Friedman proposed GBDT at 2000 [5], yet the clear convergence proof is shown in 2014 [7]. The convergence proof of Random Forest is shown in 2012 [13]. The convergence proof of AdaBoost is shown in 2013 [14].

Because of the extensive use of GBDT, the high-performance GBDT implementation is a hot topic in machine learning and high-performance computing research areas. XGBoost [15] and LightGBM [8] are generally used as the baselines for GBDT evaluations. Using GPU as the acceleration method, catBoost [10] claims 2.3x speedup with Tesla P100 versus Xeon E5-2660, and Zhang et al. [16] shows 7-8x speedup with GTX 1080 versus 2x14-core Xeon E5-2683.

Many researchers propose the algorithms to build an approximate tree that uses less computing resources to reduce the computing burden. Meng et al. [17] proposes PV-Tree, a voting approximation which avoids a large volume of data communication. pGBRT [18] first proposed to utilize histograms to speed up the creation of the decision tree in the GBDT algorithm.

Yu et al. propose PW-Tree based GBDT [19], which uses the piece-wise linear functions (which is shown in the computer is the decision tree whose leaves are linear functions) instead of piece-wise function (which is shown in the computer as normal decision trees). However, from the spect of boosting theory, that training boosting is the optimization process of function space, their work only changes the function space basis. They fail to show the convergence speed of gradient descent in piece-wise linear function space is faster than the convergence speed of gradient descent in piece-wise function space.

The standard GBDT iteration step contains two sub-steps: producing the target sub-step and building the tree sub-step. When the algorithm builds a tree, the above two sub-step should be run in sequential order. Besides standard algorithm steps [5], many studies have offered alternative and improved operations in the above two sub-steps. Sampling strategies and novel pruning strategies [20] [21] and parallelization technology on the building tree process are the main methods to accelerate the GBDT training process [8] [15].

Sampling is an alternative operation in producing the target sub-step. A number of studies have described different sampling strategies in the GBDT training process [15] [22] [20]. GBDT based on sampling [6] [23] also is named as stochastic GBDT. These stochastic GBDT algorithms focus on how to build sampling sub-datasets with good performance that represent the whole dataset well.

Some researchers have proposed a stochastic gradient boosting tree [6] and a parallel stochastic gradient boosting tree [23]. However, these methods use fork-join parallel methods. Asynchronous parallel SGD [24] [25] and its implement framework, Parameter Server [26], are widely used asynchronous parallel framework. Although Jiang et al. [27] tried to adapt the GBDT on a parameter server (i.e., TencentBoost), TencentBoost still uses the synchronous parallel method. The parallel part only exists in the sub-step of building the tree. The synchronous method fails to make full use of the performance of parameter servers.

In this paper, we examine whether it is possible to use asynchronous parallel methods to train GBDT model, how to train GBDT model in an asynchronous parallel manner, and find the conditions under which GBDT is trained in an asynchronous parallel manner effectively.

3 Background

3.1 Boosting and Its Theory

3.1.1 View GBDT and current work from probably approximately correct (PAC) learning frame

From the PAC learning aspect, the concept of GBDT can be shown in the Table 1.

GBDT in PAC Theory			
Learning model	GBDT		
hypothesis space	Function Space		
Learning method (function optimization method)	Newton Step (sequential algorithm)	Gradient Step (sequential algorithm)	Asynch-SGBDT (Parallel, this paper)
Loss function	logloss...(normally, they should be convex function)		
Implementation	XGBoost, LightGBM	LightGBM	Experimental code in this paper

Table 1: View GBDT in PAC

The function space basis for traditional GBDT is a piece-wise function, which is presented as a tree data structure in the computer. Some works use different function spaces to find the minimum of the loss function, like linear piece-wise function [19]. In the work [19], they use the linear piece function, which is the tree structure whose leaves nodes are linear functions, as a space basis to build the GBDT process. However, Newton step and Gradient step's convergence speed should be the same in different function spaces.

In Newton step and Graident step iteration, algorithms should find the next $\frac{gradient}{hessian}$ function or gradient function. However, those functions are unknown/uncomputable. We only have the output of these functions on training dataset, i.e. the training dataset's residual. Thus, we have to use the different methods to fitting these

unknown functions, i.e. building tree depending on the residual vectors.

In the building tree process, the building tree's principal algorithm is to scan the training dataset and find the best split features and the best split value. To reduce the time of building trees, many researchers propose different building tree methods: 1. The pre-sorted algorithm & histogram-based algorithm reduces the time of scanning training dataset, 2. The sampling sub-dataset reduces the burden of scanning the dataset process. 3. Split node is based on statistical value, which is used by the GOSS training method in LightGBM, 4. The voting method, which is used by the voting parallel method in LightGBM, reduces network burden. Most of these methods sacrifice the accuracy to gain a better building tree speed. These methods take effect on different levels. Thus, they can be used together.

Nowadays, high-performance GBDT implement is a hot topic. They mainly use parallel technologies like GPU [16] and distributed computing [8] to gain high performance, like XGBoost [8], LightGBM [15], DimBoost [11]. All of GBDT implements make an effect on the building tree process. Their training methods are still Newton step and Gradient step.

3.1.2 Problem Setting, Serial Training Method

Given a set of training data $\{(x_i, y_i)\}^{\sum_{j=1}^{N} m_j}$, where the feature $x_i \in \mathcal{X}$, the label $y_i = \{1, 0\}$, and m_j represents the frequency of (x_j, y_j) in the dataset, the variables (x_j, y_j) are different from each other.

The goal for boosting is to find a predictor function for the additive classifier (i.e., GBDT model), $F = F(x) \in R$, by minimizing the total loss over the training dataset

$$min_{\mathbf{F}\in\mathbb{R}^N} L(\mathbf{F}) \, , L(F) := \sum_{i=1}^{N} m_i \ell(y_i, F(x_i)). \tag{1}$$

where F_i is shorthand for $F(x_i)$ and ℓ'_i is shorthand for $\partial_{F_i}\ell(y_i, F_i)$ in the following part of the article. Theoretically, $\ell(\cdot, \cdot)$ can be an arbitrary and convex surrogate function. For example, in Friedman et al.'s work and based on the requirement in machine learning, $\ell(\cdot, \cdot)$ adopts the logistic loss:

$$\ell(y_i, F_i) = y_i log(\frac{1}{p}) + (1 - y_i) log(\frac{1}{1-p}), p = \frac{e^{F_i}}{e^{F_i} + e^{-F_i}}$$

This additive function $F(x)$ (i.e., GBDT model) produces the vector $\mathbf{F} = [F_1, F_2, ..., F_N]$ to minimize $L(\mathbf{F})$. This process is an optimization on the R^N parameter space.

We define $\mathbf{F}^*$ and $\mathbf{L}^*$ as follows.

$$\mathbf{F}^* = argmin \, L(\mathbf{F}), \mathbf{L}^* = min \, L(\mathbf{F}) \tag{2}$$

The whole process of the iteration step in traditional GBDT training is divided into two sub-steps: 1. Producing a target for the building tree. For example, the target for GBDT's Gradient step is the gradient vector of the dataset on a certain loss [7], i.e. the vector as follows:

$$\mathbf{G} = [m_1\ell'_1, m_2\ell'_2, ...m_N\ell'_N] \tag{3}$$

2. Building a tree whose output is close to the target. For example, in GBDT Gradient step, the prediction for x_i is close to ℓ'_i in this tree.

Building a decision tree algorithms have ID3, C4.5, CART. For tree building, GBDT frameworks, like LightGBM and XGBoost, commonly use the algorithm of pre-sort and histogram algorithm. Our method focuses on how to build different trees among different workers in parallel. Within each worker, we also use pre-sort and histogram algorithm for tree building step.

3.1.3 GBDT Training Method

Based on the work of Sun et al. [7], GBDT can be trained by Newton step and Gradient step. Current GBDT implementations normally use Newton step since it demonstrates relatively fast serial convergence, like XGBoost. LightGBM can work in Newton step and Gradient step both, which depends on "is_constant_hessian" attribute.

The convergence speed of Newton step, which XGBoost uses, is faster than Gradient step. However, Newton step requires the $\ell(\cdot)$ has a Hessian matrix. By contrast, Gradient step does not require the mathematical property, but the convergence speed is slow. The classical $\ell(\cdot)$s which have to be trained by Gradient step in LightGBM are the L2 loss for regression task("RegressionL2loss" choice in LightGBM), Mape loss for regression task("RegressionMAPELOSS" choice in LightGBM), Huber loss for regression task("RegressionHuberLoss" choice in LightGBM) and so on. In a word, Newton step and Gradient step are used in different cases. Gradient step training method has wide application scope, and Newton step training method is fast.

3.1.4 Parallel GBDT Training Algorithm

Current GBDT algorithm uses fork-join parallel implementation to parallelize the GBDT training process. Only the parallel parts stay in the building tree sub-step process in the current GBDT framework and algorithm, such as XGBoost [8] and LightGBM [15].

3.2 GBDT trees

Decision trees are an ideal base learner for data mining applications of boosting (Section 10.7 in the book [28]). Thus, this subsection, we would present the necessary theories of the decision tree.

The goal of building a decision tree is that each leaf in tree reach the minimum RMSE or misclassification rate.

Above statement in GBDT case is that for the decision tree, the prediction for x_i is ℓ'_i in Eq. 3. Without regard to generalisation, a well-grown decision tree often reaches above goal.

Above statements are also equivalent to the statement that the target of the process of building tree is to build the tree whose leaves contain the almost the same samples.

3.3 SGD and Parameters Server

SGD is used to solve the following problem [29]

$$minimize_{w\in X} f(w) \, , f(w) := E\left[Function(w; \Theta)\right] \tag{4}$$

$$E\left[Function(w; \mathbf{\Theta})\right] = \int_{\Xi} Function(w; \theta) dP(\theta) \tag{5}$$

where a random variable $\mathbf{\Theta}$ has a probability distribution function (PDF) $dP(\theta), \theta \in \Xi$. We use the frequency instead of $dP(\xi)$, which means

$$f(w) \approxeq \frac{1}{n} \sum_{1}^{n} Function(w, \theta_i) \tag{6}$$

where θ_i is the observed value of random variable $\mathbf{\Theta}$. A dataset consists of θ_is.

The algorithm of the delayed SGD (i.e., an asynchronous parallel SGD algorithm) [24] [25] is described as an important parallel SGD. The implements of delayed SGD, (parameter server) include ps-lite in MXNET [30], TensorFlow [31], and petuum [32].

Random variable In a traditional machine learning problem, such as training a support vector machine (SVM) model, an observed value θ_i of random variable $\mathbf{\Theta}$ is a sample in the dataset.

3.4 Sampling and Stochastic GBDT

Many studies have also proposed the stochastic GBDT algorithm which uses sampling strategies. GBDT algorithms based on sampling datasets are also named as stochastic GBDT algorithms [6] [23]. However, the above studies fail to involve asynchronous parallel methods. They use sampling strategies to reduce the burden of building a tree to improve the accuracy of the GBDT output [15] [22] [20]. These works focus on how to make sample sub-datasets that share the same characteristics with the full dataset.

4 Main Idea

As we can see from the above section, almost only SGD and its related algorithms have asynchronous parallel training methods. Thus, we have a basic idea to build a stochastic optimization problem which shares the same output with an original GBDT training problem. Stochastic GBDT, which uses sampling strategies, rightly offered this stochastic optimization.

Corollary 1 The mathematical expectation of output of training stochastic GBDT and training GBDT are the same.

Proof. In stochastic GBDT, each iteration step is described as follows: 1. Sampling the data: Each sample in the dataset corresponds to a Bernoulli distribution. In each iteration step, selecting this sample depends on its Bernoulli distribution. Traditional stochastic GBDTs view sub-datasets and full datasets share the same characteristics. 2. Build target: In Gradient step, stochastic GBDT calculates the gradient of a sub-dataset on a certain loss. 3. Building the tree based on the target.

However, we treat the sampling process as the method which introduces a random variable into the original objective function. Then, the goal for stochastic GBDT is to find an additive function $\mathbf{F}$ by minimising the mathematical expectation of the total loss over the training dataset.

$$
\begin{aligned}
&minimize_{\mathbf{F}\in\mathbb{R}^N} f(\mathbf{F}) \, , f(F) := E\left[L_{random}(\mathbf{F};\mathbf{Q})\right] \\
&L_{random}(\mathbf{F};\mathbf{Q}) := \sum_{i=1}^{N}(\sum_{j=1}^{m_j}\frac{Q_{i,j}}{R_{i,j}})\ell(y_i,F_i) \qquad (7)
\end{aligned}
$$

where $\mathbf{Q} = (Q_{1,1}..Q_{1,m_1}, Q_{2,1}, ..., Q_{2,m_2}, Q_{N,1}, ..., Q_{N,m_N})$ and $Q_{i,j}$ is a random variable that satisfies the Bernoulli distribution: $P(Q_{i,j}=1) = R_{i,j}$, $P(Q_{i,j}=0) = 1 - R_{i,j}$.

In every sampling process, the sampling operation in the stochastic GBDT would produce an observed value vector corresponding to $\mathbf{Q}$. Based on the observed value vector, the sampling operation in stochastic GBDT produces the sampling sub-dataset.

Combining the convex characteristics of L and L_{random}, the following expressions are true.

$$
min\, E[L_{random}] = min\, L \qquad (8)
$$

$$
argmin\, E[L_{random}] = argmin\, L \qquad (9)
$$

Now, our optimization objective function is changed from $L(\mathbf{F})$ to $L_{random}(\mathbf{F};\mathbf{Q})$, which means training stochastic GBDT and stochastic optimisation are the equivalence problem. □

Using SGD to Solve Stochastic GBDT Corollary 1 shows that in mathematical form, training stochastic GBDT and stochastic optimization algorithm are the same. Eq. 7 shows that $\mathbf{Q}$ is a random variable vector and $\mathbf{F}$ is a variable. The definition of L_{random} in Eq. 7 is the same as the expression of $Function(w;\mathbf{\Theta})$ in Eq.4. The above facts suggest that it is possible to use well parallelised stochastic optimisation, such as the asynchronous parallel SGD algorithm, to find the minimum of L_{random}.

Thus, we have a starting point to use high-performance stochastic optimization algorithm, like asynchronous parallel SGD to train stochastic GBDT. In this paper, we name it as asynch-SGBDT.

Basic proof structure of asynch-SGBDT is as follows: 1. estimate the difference between L'_{random} and the output of decision tree which is based on L'_{random}. 2. Add above difference into asynchronous SGD proof structure and see how this difference influences convergence speed and convergence point. 3. Focus on the coefficient before the delay(which is the parallel degree), to see what factors influence the scalability. The whole proof is shown in the appendix and in body of paper, to help reader understand easily, we show a analysis of a special case.

Random Variable In traditional machine learning problems, such as training an SVM model, an observed value of a random variable is a sample in the dataset. In asynch-SGBDT, an observed value of a random variable is the sampling result of the sampling sub-dataset.

5 Asynch-SGBDT and its Analyses

5.1 Asynch-SGBDT

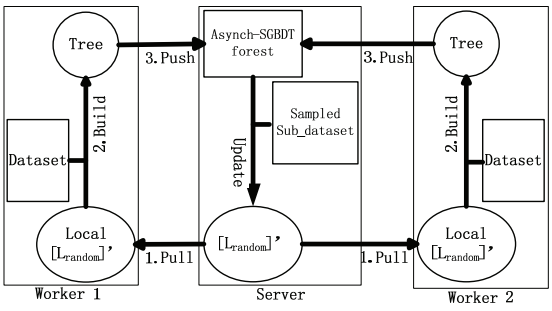


Figure 1: Asynch-SGBDT on the parameter server**:** worker 1 and worker 2 are asynchronously parallel and independently work. The server updates L'_{random} at once when it receives a tree from any worker.

Asynch-SGBDT uses asynchronous parallel SGD to train the model. Asynch-SGBDT is described by algorithm 1. The work model of asynch-SGBDT is illustrated in Figure 1. The sequence diagrams of asynch-SGBDT are shown in the last sub-figure in Figure 2. In Figure 2, we also show the sequence diagrams of the different GBDT training methods.

In algorithm 1, L'_{random} is shorthand for the stochastic gradient of $L_{random}(\mathbf{F};\mathbf{Q})$ and L'_{random} is calculated as follows

$$L'_{random} := [m'_1\ell'_1, m'_2\ell'_2, ..., m'_N\ell'_N] \tag{10}$$

where $m'_i = \sum_{j=1}^{m_i} \frac{Q_{i,j}}{R_{i,j}}$. In building the tree sub-step, the algorithm still builds the tree whose prediction for x_i is close to ℓ'_i.

Asynchronous parallel Asynchronous parallel in this context means that different workers conduct their work independently to each other. These workers work independently. The pull, build, and push operations of a worker must be ordered and serialised but, for different workers, these operations in different workers are completely out of order and parallel. During the time that a worker is building the ith tree, $F^t(x)$ and L'^t_{random} in the server would be updated several times by other workers. If the number of workers is large enough, the building tree sub-step would be hidden. Additionally, compared with the original datasets, the size of the sampling sub-dataset is relatively small, which would reduce the burden of the building tree sub-step.

Algorithm 1 Asynch-SGBDT

Input: $\{x_i, y_i\}^N$: The training set; v: The step length;
Output:the Additive Tree Model $F = F(x)$, i.e. asynch-SGBDT model.
For Server:
Produce the tree whose output is $\frac{1}{\sum_{i=0}^{N} m_i}\sum_{i=0}^{N} m_i y_i$.
Calculate L'^0_{random} and Maintain L'^0_{random}.
for $j = 1...forever$ **do**
1.Recv a $Tree_{k(j)}$ from any worker, this tree is built based on $L'^{k(j)}_{random}$.
2.Add $Tree_{k(j)}$ times v to whole asynch-SGBDT model. ($F^j(x) = F^{j-1}(x) + vTree_{k(j)}$)
3.Generate an observed value vector of $\mathbf{Q}$ and produce sampling sub-dataset.
4.Calculate current GBDT model's L'^j_{random} based on sampling sub-dataset.
5.Remove L'^{j-1}_{random} and Maintain L'^j_{random} (L'^j_{random} can be pulled by workers.).
end for
For Worker:
for $t = 1$ to forever **do**
1.Pull the L'^t_{random} from Server (L'^t_{random} is current L'_{random} the Server holds.).
2.Build $Tree_t$ based on L'^t_{random}.
3.Send $Tree_t$ to Server.
end for

5.2 Convergent analysis of Asynch-SGBDT for Special Case

To make our presentation clear, we show a result of a special case which would show how different variables work intuitively without paying too much cost. General case and conclusion will offer in the next subsection.

Our analysis in this subsection should be based on the assumption that each decision tree in GBDT is a well-grown tree, which is equivalent to the case that the build the tree whose leaves contain the almost the same samples. More information is provided in section 3.2.

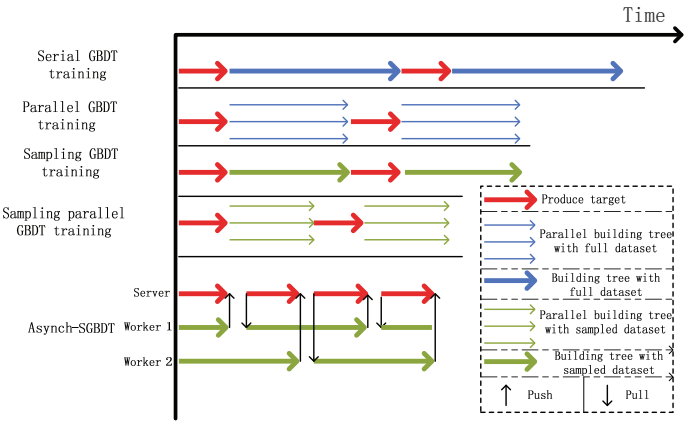


Figure 2: Different GBDT Training Method Patterns

We introduce additional math definitions as follows: Lip is the Lipschitz constant for $L(.)$. $L(.)$ is a strong convex function with modulus c. For $\mathbf{F} \in \mathbb{R}^N$ and every observed value vector corresponding to $\mathbf{Q}$, $\|L'_{random}\| < M''$. The random variable vector $\mathbf{Q}'$ is defined as

$$\mathbf{Q}' = [Q'_1, Q'_2, ..., Q'_N]$$

where the random variable $Q'_i = (Q_{i,1} \vee Q_{i,2} \vee ... \vee Q_{i,m_i})$. $\Omega' = max\sum_{i=1}^{N} Q'_i$, which represents the maximum number of different samples in one sampling process. $\Delta = max\, P(Q'_i = 1)$, which is the maximum probability that a type of sample is sampled. ρ' represents the probability that two sampling sub-datasets, whose intersection is nonempty, exist throughout the whole sampling process.

We apply the proposition of asynchronous SGD [25] to Algorithm 1.

Proposition 1. Suppose in Algorithm 1 that
(1) $\tau \geq j - k(j)$, (2) v is defined as

$$v = \frac{c\vartheta\epsilon}{2LipM'^2\Omega'(1 + 6\rho'\tau + 4\rho'\tau^2\Omega'\Delta^{1/2})} \tag{11}$$

(3) By defining $D_0 := \|\mathbf{F}_0 - \mathbf{F}^*\|^2$, $\mathbf{F}_t$ is the $\mathbf{F}$ vector produced by $F^t(x)$. (4) For some $\epsilon > 0$ and $\vartheta \in (0, 1)$, t is an integer satisfying

$$t \geq \frac{2LipM'^2\Omega'(1 + 6\rho'\tau + 6\rho'\tau^2\Omega'\Delta^{1/2}log(LD_0/\epsilon))}{c^2\vartheta\epsilon} \tag{12}$$

Then after t updates in the server,

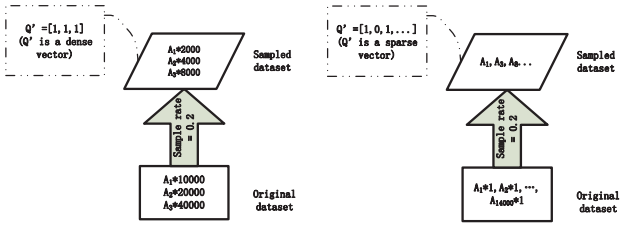
$$E[L(\mathbf{F}_t) - L^*] < \epsilon.$$

5.2.1 *Analysis: Convergence Speed*

Proposition 1 shows that stochastic GBDT converges to a minimum of $L(\cdot)$. Proposition 1 also shows that with the increase in the number of workers (i.e., the number of delays), a smaller step length should be chosen, and asynch-SGBDT needs more iteration steps to reach a satisfactory output.

5.2.2 *Analysis: Scalability Upper Bound*

It is easy to see that the max number of worker algorithm satisfies following upper bound.



(a) The original dataset contain samsamples: 10000 * A_1, 20000*A_2 30000*A_3. The sample diversity of orighinal dataset is low

(b) The original dataset contain samples: A_1 ... A_{14000} and each sample only appears once. The sample diversity of orighinal dataset is high

Figure 3: The sample diversity in dataset exert influence on the the sparsity of $\mathbf{Q}$

$$\#(worker) < \frac{T(BuildTree)}{T(Communicate + BuildTarget)} \tag{13}$$

where $\#(worker)$ is the max number of workers, $T(operation)$ is the time of operations. This inequality can be obtained from the condition that computing time for each worker is fully overlapped from the computing time in the server.

5.2.3 Analysis: Scalability and Sensitivity

The sensitivity of the mathematical convergence speed to the change in the number of workers is another index that measures the scalability of the algorithm. If the mathematical convergence speed is insensitive to the number of workers (i.e., parallelism), the algorithm allows us to use more workers to accelerate the training process and gain a greater speedup.

Our proposition shows that sensitivity is linked to ρ' and Δ. The sparsity of the observed value vector of $\mathbf{Q}'$ is positively correlated to the values of ρ' and Δ when sampling rates between different samples are almost the same or uniform. Therefore, it is possible to offer guidance and draw conclusions by analyzing the observed value vector of the sparsity of $\mathbf{Q}'$.

The size of $\mathbf{Q}'$ represents the diversity of the samples in the dataset (i.e., the number of sample species). The number of non-zero elements in each sampling process represents the diversity of the samples in the sampling sub-dataset. If the diversity of the dataset is high, but the diversity of the sampling sub-dataset is low, the observed value vector of $\mathbf{Q}'$ is likely to be a sparse vector in each sampling process, which means that the values of ρ' and Δ would be small. The algorithm would be insensitive to the number of workers. Reducing the sampling rates would help reduce the diversity of the sampling sub-dataset. Using a high-dimensional sparse dataset would contribute to increasing the diversity of the dataset. For the GBDT tree, different samples would be treated as the same sample if the number of leaves is too small. In this case, a small number of leaves in the GBDT tree would decrease the diversity of the dataset. For a low diversity dataset, even using the small sampling rate, the observed value vector of $\mathbf{Q}'$ is still a dense vector, just like the illustration of Figure 3. Therefore, asynch-SGBDT is able to accelerate the high-dimensional sparse dataset with a relatively small sampling rate. The high-dimensional sparse dataset is frequently used dataset in the era of big data.

5.3 Convergent analysis of Asynch-SGBDT for General case

5.3.1 Iteration Step

To make our analysis more general, we will use the following iteration step to describe our algorithm

$$\begin{aligned}\mathbf{F}^{j+1} &= \mathbf{F}^j - vV(V^TPV)^{-1}V^TL'^{k(j)}_{random} \\ &= \mathbf{F}^j - vV(V^TPV)^{-1}V^TPL'^{k(j)}_{base}\end{aligned} \tag{14}$$

where $L_{base} = (\ell_1, \ell_2, ..., \ell_N)$. The matrix V is defined as the projection matrix in Sun's work [7] and P is the diagonal random variable matrix: $P = diag(\sum_{j=1}^{m_1}\frac{Q_{1,j}}{R_{1,j}}, \sum_{j=1}^{m_2}\frac{Q_{2,j}}{R_{2,j}}, ..., \sum_{j=1}^{m_N}\frac{Q_{N,j}}{R_{N,j}})$ and $P_{fix} = diag(m_1, m_2, ..., m_N)$. Thus, $\mathbb{E}P = P_{fix}$

We can summarize the following relationship:

$$P_{fix}L'_{base} = \mathbf{G} = \mathbb{E}PL'_{base} = \mathbb{E}L'_{random}$$

In following presentation, we let $A = V(V^TPV)^{-1}V^T$.

5.3.2 New notation

To help readers have a clear comparison with special case, we define some new notations and redefine some old notes in the body of paper. In following part of paper and appendix, all notes are based on this subsection.

λ is the Lipschitz constant for $\ell(.)$. $L(.)$ is a strong convex function with modulus c. For $\mathbf{F} \in \mathbb{R}^N$ and every observed value vector corresponding to $\mathbf{Q}$, $\|AL'_{random}\| < M$.

Ω represents the maximum number of different non-zero component for the vector AL'_{random}.

ρ represents the probability that two vector multiplication of AL'_{random} is zero.

ζ represents the maximum number of non-zero component for the vector $AL'_{random} - L'_{random}$. This value can measure the tortuosity of decision tree to L'_{random}. Apparently, when the tree has more leaves, ζ is close to zero.

$m_{max} = \max\{m_1, m_2, ..., m_N\}$.

5.3.3 Assumption

It is normal to use following assumptions.

c-strong convex The $f(\cdot)$ is c-strong convex function, i.e.

$$f(F^{j+1}) - f(F^j) + \frac{c}{2}\left\|F^{j+1} - F^j\right\|^2 \leq (F^{j+1} - F^j)^T f'(F^{j+1}) \tag{15}$$

Let F^* be the minimal of ℓ, and we have

$$\frac{c}{2}\left\|F^j - F^*\right\|^2 \leq (F^j - F^*)^T f'(F^j) \tag{16}$$

In our paper, $L(\mathbf{F})$ and $\ell(\cdot, \cdot)$ are c-strong convex function.

λ-Lipschit function The $f(\cdot)$ is λ-Lipschit function, i.e.

$$\left\|f'(F^{j+1}) - f'(F^j)\right\| \leq \lambda\left\|F^{j+1} - F^j\right\| \tag{17}$$

In our paper, $\ell(\cdot, \cdot)$ are λ-Lipschit functions.$L(\mathbf{F})$ are $m_{max}N$-strong Lipschit smoothness functions.

Bound of gradient's norm In this paper, we need to set the bound of gradient's norm.

$$\|\ell'\| \leq \phi$$

Thus, we define the bound of gradient of $L(\cdot)$, M, and we gain the following equations.

$$\Omega m_{max}^2 \phi^2 \geq M^2 \geq L_{base}'^T P^T A P L_{base}'$$

The sample's similarities in one GBDT's leaf For the decision tree is the classifier which classify sample via their space position. For the samples $\{(x_i, y_i)\}^{\sum_{j=1}^{N} m_j}$ in one leaf, we have following equation:

$$\max_{i,j \in leaf_k} \|x_i - x_j\| \leq \delta \tag{18}$$

5.3.4 Convergence conclusion

The convergence proof of asynch-SGBDT is shown in Appendix section. Based on our analysis and proof in appendix, we can gain the following Proposition.

Proposition 2 The asynch-SGBDT is trained on Parameter Server under the assumptions above subsection mentioned, then

$$\begin{aligned}
&\mathbb{E}\left\|\mathbf{F}^{j+1} - \mathbf{F}^*\right\|^2 - \mathbb{E}\left\|\mathbf{F}^{\infty} - \mathbf{F}^*\right\|^2 \\
&= r\left(\mathbb{E}\left\|\mathbf{F}^{j} - \mathbf{F}^*\right\|^2 - \mathbb{E}\left\|\mathbf{F}^{\infty} - \mathbf{F}^*\right\|^2\right) \\
&\mathbb{E}\left\|\mathbf{F}^{\infty} - \mathbf{F}^*\right\|^2 = \left(\frac{C_1 + \sqrt{C_1^2 + 4cvC_2}}{2c}\right)^2 \qquad (19) \\
&r = 1 - vc\left(1 - \frac{C_1}{C_1 + \sqrt{C_1^2 + 4cvC_2}}\right) \qquad (20)
\end{aligned}$$

where C_1, C_2 are defined as :

$$\begin{aligned}
C_1 &= (2\delta\lambda m_{max}\sqrt{\zeta} + cv\tau M) \\
C_2 &= (4\delta\lambda m_{max}\sqrt{\zeta} + cv\tau M)\tau M + 2M^2(3\rho\tau + \frac{1}{2})
\end{aligned}$$

5.3.5 General Conclusion

We summarize the general conclusions as follows:

1. Small sampling rate would decrease the convergence speed. However, small sampling rate help decrease the sensitivity of the algorithm, which would increase the scalability.

2. Under an asynchronous parallel situation and using a fixed sampling rate, the more workers we use, the smaller the step length that should be chosen, and the more steps the algorithm would run.

3. Under an asynchronous parallel situation and using a fixed number of workers, the larger the sampling rate is, the more sensitive the algorithm.

4. Under an asynchronous parallel situation and using a fixed number of workers, the smaller the sampling rate is, the larger the step length that should be chosen, and the fewer steps the algorithm would run.

5. Asynch-SGBDT is apt to accelerate the GBDT on a high-dimensional sparse dataset.

6. Asynch-SGBDT is apt to accelerate a GBDT whose trees contain massive leaves.

We also can draw the requirements of high scalability for asynch-SGBDT: high dataset diversity, small sampling rate and large learning rate, large GBDT leaves number setting. Those requirements are named as asynch-SGBDT requirements in this paper. Usually, normal GBDT settings on large scale dataset satisfy asynch-SGBDT requirements.

Besides the above general conclusions, we also can draw the following counter-intuitive conclusions: 1. The sampling process is necessary for asynchronous parallel; even we have enough computing resource. 2. Only Gradient step can use asynchronous parallel manner. Thus, XGBoost cannot be modified into asynch-parallel manner.

6 EXPERIMENT

The experiment part consists of two part: validity experiments part and efficiency experiments part. Validity experiments are designed to prove that asynch-SGBDT can convergence to minimum without slowing down convergence speed on the epoch. Efficiency experiments are designed to prove that asynch-SGBDT is faster than traditional synchronous parallel manner.

Validity experiments: To prove the algorithm validity and correspond with the asynch-SGBDT theoretical analyses, we offer validity experiments. In the experiments, we used asynch-SGBDT to deal with the two classification problems.

Efficiency experiments: To prove the asynchronous parallel method is better than the synchronous parallel method, we compared the speed up ratio of asynch-SGBDT with feature parallel. Feature parallel is the main parallel method in LightGBM [15].

Extra experiments: To make the sampling conclusion clearly, we conduct HIGGS experiments. We also offer an easy conducted corollary experiment, to reveal a profound relationship between sequential GBDT training process and asynch-SGBDT.

6.1 Code Configuration

Our experiment codes are modified from LightGBM.

The first step for this modification is changing LightGBM into Gradient step, which is changing "is_constant_hessian" attributes in LightGBM's GBDT class into true.

The second step for this modification is to migrate GBDT training process into ps-lite in MXNet: Firstly, in Server and Worker initialize the application class with the same setting. Secondly, Server produce "gradient" array based on our algorithm and send it to worker. Thirdly, Workers pull "gradient" array from Server, put the "gradient" array into "TrainOneIteration" method, and send the tree string (which is used "ToString" method in tree class) to Server. Finally, Server receive tree string, recover tree class, push tree class into "model_" attributes in GBDT class and update "gradient" array.

Our codes share the same code with LightGBM with one worker, just shown in figure 4.

6.2 Benchmark

From the view of PAC learning frame, the GBDT training process is the optimization process in function space. There are only two methods to train a GBDT model: Gradient Step and Newton Step. They are all sequential algorithms.

XGBoost is the implementation of Newton step, and LightGBM is the implementation of Newton step and Gradient step. Their main contribution is different building tree methods. Building tree methods do not influence the essence of training methods. If a work mainly talks about how to build a tree using any algorithms like sampling/histogram, the work's contribution is the building tree method instead of the training method. Building tree algorithms are not the same level as Gradient step/Newton step/asynch-SGBDT. Gradient step and Newton step are used in different cases. Our

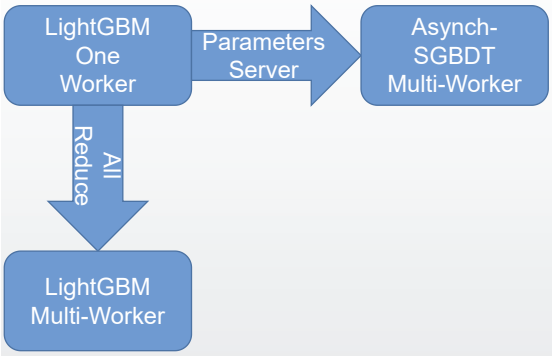


Figure 4: Experiment code and LightGBM with multi worker share the same codes when the number of worker is one

work is based on Gradient Step. Thus, it is suitable to use Gradient Step as a benchmark.

Our efficiency experiment benchmark is LightGBM in Gradient step. LightGBM is reported as state-of-the-art high-performance parallel GBDT implementation [9]. These benchmarks will strongly show that our algorithm is better than synchronous parallel GBDT algorithms.

For LightGBM has three parallel models: top-k parallel model, data-parallel model and feature parallel model. From the view of scalability and accuracy, the feature parallel is the best choice of the benchmark: The network load will be increased with the increase of the number of workers in data-parallel. The accuracy of the top-k parallel model is worse than the standard GBDT train method [15].

6.3 Datasets

We chose four datasets: the real-sim, KDD2010, KDD2012 and HIGGS datasets, which were selected from the SVM library (LIBSVM) repository as our experimental dataset. real-sim, KDD2010 and KDD2012 datasets are high dimension sparse datasets whose sample diversity is usually high. Table 2 shows all dataset information.

dataset	#sample train	#sample test	#feature	density
real-sim	60000	12000	20958	$< 1\%$
KDD2010 (part)	350000	70000	20216830	$< 1\%$
KDD2012 (part)	450000	90000	54686452	$< 1\%$
Criteo (part)	400000	40000	1000000	$< 1\%$
webspam (part)	35000	6000	16609143	$< 1\%$
HIGGS (bad case)	100000	20000	28	100%

Table 2: Dataset information

Because single machine cannot use full data from HIGGS, KDD2010 and KDD2012, we use part of them as our training data.

HIGGS dataset does not satisfy the requirements that asynch-SGBDT requirements (High sample diversity/High dimension and sparse). Thus, we use it as a bad case and make comparison experiments in sensitivity and scalability experiment for bad case experiments can show many interesting characters (High sample diversity dataset experiments do not exhibit these characters clearly).

6.4 Experiment Algorithm Settings

For different datasets have different algorithm setting, we summarize them in the following table 3.

dataset	# leaf	learning rate	# tree	min data in leaf	feature fraction
real-sim	400	0.01	400	5	100%
KDD2010 (part)	1000	0.01	10000	15	80%
KDD2012 (part)	1000	0.001	10000	15	75%
Criteo (part)	2000	0.001	5000	10	100%
Webspam	200	0.0001	5000	10	100%
HIGGS(bad case)	20	0.01	1000	5	100%

Table 3: Experiment Algorithm Settings

In different experiment settings, sampling rates are different in different experiments, which we will show it in experimental results. It is worth noting that in the following experimental results, we only offer the data when GBDT reach its convergence point.

6.5 Experiment Environment Settings

Our experiments are runs on a modest cluster with 32 nodes. In this cluster, each node consists of Intel E5-2680 V2 CPU and 64 GB memory. The cluster uses Cent OS 6.4 and gcc 4.4.7. All nodes are connected by Intel(R) I350 Gigabit Network Connection. Because of the limitation of cluster's scale, in the experiments which use 1,2,4,8,16,32 nodes, one worker occupied one server and in the experiments which use 64, 128 nodes, two or four workers are addressed in one server.

Although our environment has Infiniband as their connection, we have to only use Intel(R) I350 Gigabit Network Connection because ps-lite in MXNet using ZeroMQ as their connection tool and ZeroMQ cannot be used in Infiniband. Ps-lite uses ZeroMQ for the communication between worker and server, and LightGBM uses socket for the communication of AllReduce.

For 2/4 LightGBM workers have to be addressed into one server in some experiments, to reduce the resource competition in a node, we are setting each LightGBM worker uses one OpenMP thread.

6.6 Validity experiment

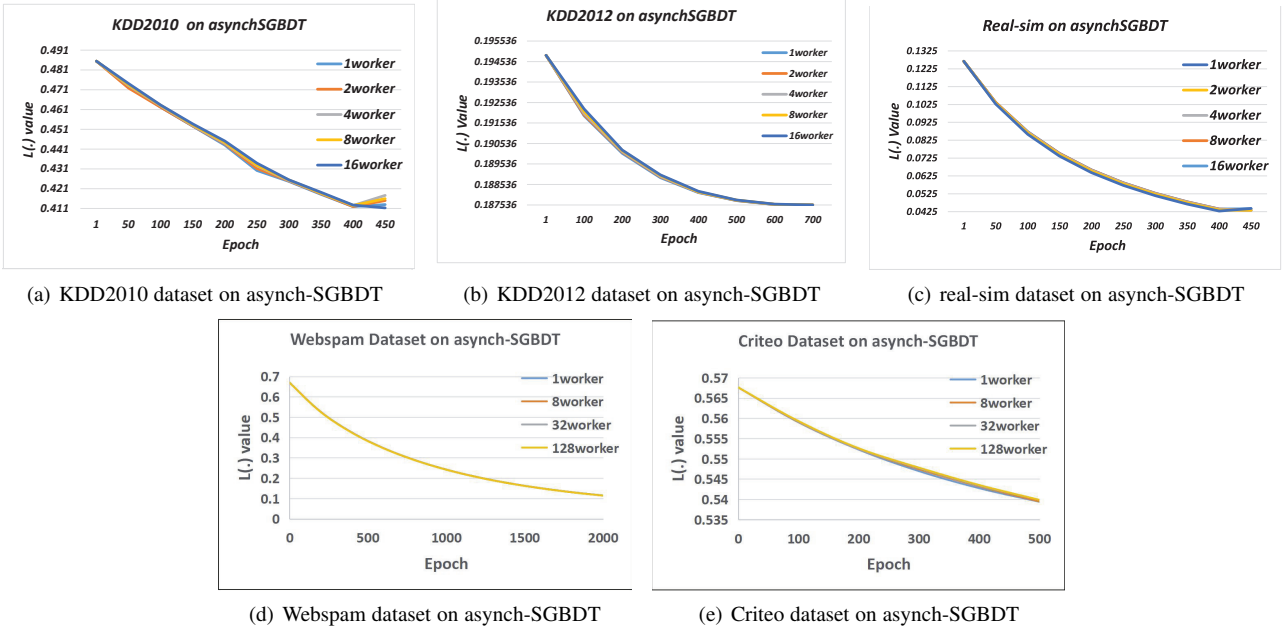
6.6.1 Experimental Results and Analysis - Convergence Experiment

In this subsection, we will show that asynch-SGBDT is able to work on different high dimension sparse datasets.

dataset	num_tree	LogLoss: 1worker(Sequential Algorithm)	LogLoss: 16workers
real-sim	400	0.04408	0.044282
KDD2010(part)	400	0.41199	0.4127
KDD2012(part)	700	0.187536	0.187618
Criteo(part)	500	0.539376	0.539820
Webspam	2000	0.114217	0.114309

Table 4: Comparison of accuracy for different number of worker

The convergence experiment results are shown in figure 5 and the accuracy comparison is shown in table 4. In this experiment, we can get the following conclusions: 1, when datasets and algorithm



(a) KDD2010 dataset on asynch-SGBDT (b) KDD2012 dataset on asynch-SGBDT (c) real-sim dataset on asynch-SGBDT

(d) Webspam dataset on asynch-SGBDT (e) Criteo dataset on asynch-SGBDT

Figure 5: Asynch-SGBDT on different datasets with sampling rate 0.2

settings satisfy general conclusions in 5.3.5 subsection, the number of workers will show little influence on convergence speed on epoch, i.e., the number of trees. 2, under the fixed sampling settings, the more workers we use, the slightly slower the convergence speed. These experimental results are described by conclusion 2 in section 5.3.5, where the more workers we use, the slower the convergence.

6.6.2 Experimental Results and Analysis - Sensitivity and Sampling rate Experiment

In this subsection, we will show how the sampling rate influences sensitivity and scalability. In this subsection, we will focus on the dataset: real-sim, KDD2012, Webspam, Criteo and HIGGS.

HIGGS is a bad case to make comparison experiment.

The real-sim, Webspam, Criteo and KDD2012 experimental results are shown in Figures 6. Figure 6 shows that under the fixed sampling settings, the more workers we use, the slightly slower the convergence speed. Experimental results show that the speedup rises linearly with the increase in the number of workers in unlimited network resource condition. These experimental results are described by conclusion 2 in section 5.3.5, where the more workers we use, the slower the convergence.

Above figures show that when datasets are high dimension and sparse, sampling rates between 0.2 and 0.8 exert a slight effect on the convergence speed in this dataset. However, the sampling rate will partially influence the convergence point.

We also conducted an experiment using an extremely small sampling rate (sampling rate = 0.000005, which means we use approximately 500 samples on average in each sampling sub-dataset). The baseline experiment uses a normal sampling rate (sampling rate = 0.6). The result is shown in (b) subfigure in Figure 8. The small sampling rate, which produces a small sub-dataset, reduces the sample diversity in the sampling sub-dataset. Small sample diversity in the sampled dataset would help reduce Δ and ρ. This experiment shows that a small sampling rate would help reduce the sensitivity with the help of reducing Δ and ρ: The gap between different convergence lines in small sampling rate experiment is smaller than it in large sampling rate experiment.

However, an extremely small sampling rate would decrease the convergence speed because the sub-dataset is too small, which would cause the GBDT trees to be distorted. The experimental results in Figure 8 match conclusions 1 and 3 in section 5.3.5.

6.7 Efficiency experiments

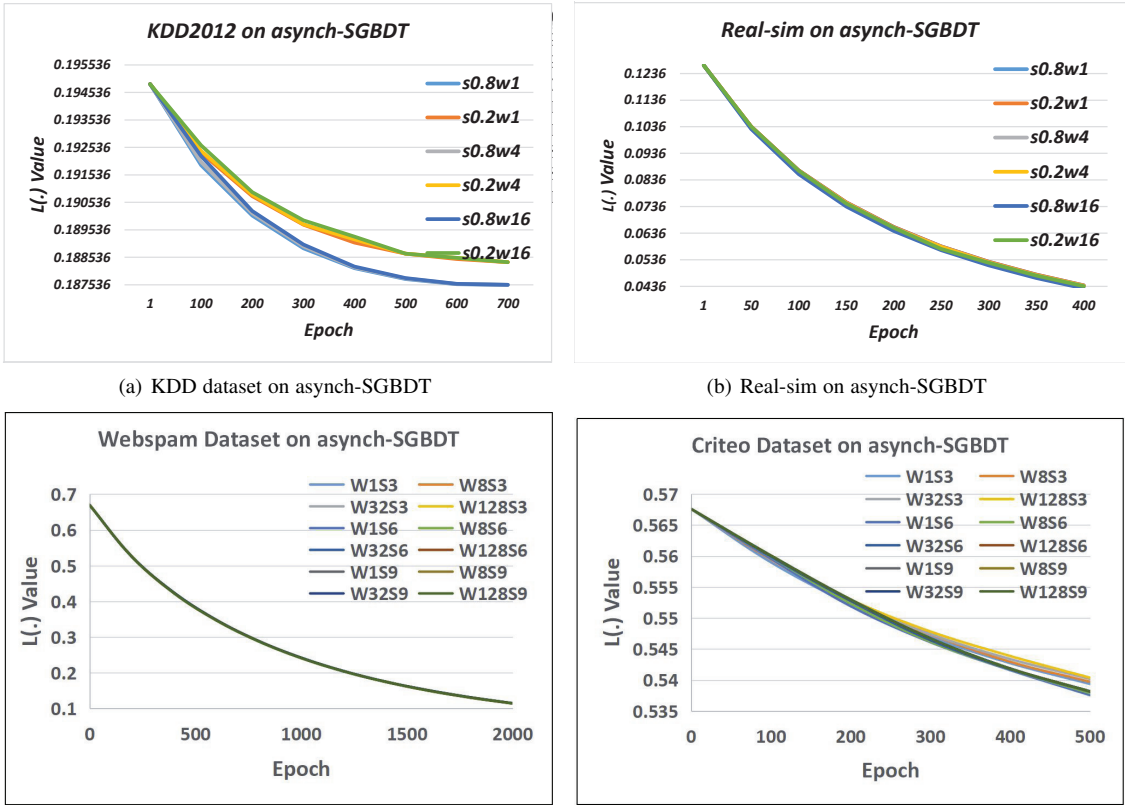
6.7.1 Experiment Settings

In this section, we will show that the asynchronous parallel method, like parameters server, is more suitable to current parallel computing environment than the synchronous parallel manner, like fork-join or MapReduce parallel method.

6.7.2 Analysis of Experimental Results

Figure 7 shows the speedup and time between asynch-SGBDT and LightGBM. Our experiment shows that in real-sim dataset experiment, asynch-SGBDT achieves 14x speedup using 32 nodes. In KDD2010 and KDD2012 dataset, compared with LightGBM which uses one worker, asynch-SGBDT achieves 11x-12x speedup using 32 nodes. However, feature parallel method used in LightGBM is not effective as asynch-SGBDT: in the real-sim, Webspam, Criteo, KDD2010 and KDD2012 datasets, compared with LightGBM which uses one worker, LightGBM achieves 6x-7x speedup using 32 machines.

Asynch-SGBDT and LightGBM show a great difference in speedup. Especially with the increase of the number of machines or workers, the gap is expanded. This phenomenon is caused by 1. The node performance inconsistency in a cluster. In the synchronous parallel algorithm, the algorithm does not continue the process until all nodes reach the barrier. The increase of the number of machines would exacerbate the cost of synchronous operation. 2. Asynch-SGBDT is robust against network instability and the computing time and communication time is overlapped in asynch-SGBDT. 3. In Asynch-SGBDT, whole building tree process is running parallel.



(a) KDD dataset on asynch-SGBDT

(b) Real-sim on asynch-SGBDT

(c) Webspam on asynch-SGBDT

(d) Criteo on asynch-SGBDT

Figure 6: Asynch-SGBDT with different sampling rates on different datasets(s-sampling rate, w-the number of worker)

In LightGBM, only collection feature information process can be implemented in a parallel method. Collection feature information process is just a part of build tree process.

Asynch-SGBDT experiments using different datasets also show a great difference in speedup. This phenomenon is caused by the network problem: In Webspam, Criteo, KDD2012 and KDD2010 dataset experiment, the ZeroMQ network has to load huge data transfer task, which cost a lot of time. Eq.13 shows that in this case, the upper bound of the max number of the worker in asynch-SGBDT would be reduced. In Asynch-SGBDT, we still cannot reach the linear speedup for the following reasons: 1. the parameters server's limitation: the parameters server has the computation load, i.e., update the L'_{random}. This work can be fast, but in our experiment setting, parameters server only uses one thread to dealing with the whole update work, which creates the bottleneck. 2. The performances of nodes are roughly the same, and their work are started at almost the same time. Thus, during the training process, different workers are submitting their trees at a short time but the server have to dealing with update target work sequentially, which cases the bottleneck.

6.8 Extra Experiments

6.8.1 HIGGS Experiments

In real-sim, Webspam, Criteo, and KDD2012 experiments, the influence of the sampling rate is small for those datasets are so fit for asynch-SGBDT. To present sampling rate has a great influence on sensitivity, we do asynch-SGBDT experiments on HIGGS dataset.

Based on the theory, which we offered in the appendix when the GBDT trees have a few leaves, the influence of sampling would be remarkable. Thus, we conduce HIGGS dataset experiments for the max leaves is 28 in this experiment.

The experimental results are shown in figure 8.

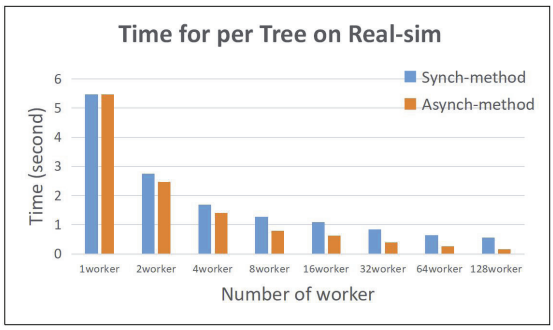
In the HIGGS experiments, subfigure (a) in figure 8, the dimension of the sample vector and the range of a feature value are relatively small, which leads to diversity in the HIGGS dataset being relatively small. From a mathematical perspective, this result means that the observed value vector of $\mathbf{Q}'$ is almost equal to $[1, 1, ..., 1]$ in every sampling process. Δ and ρ would be large to this situation. Additionally, the number of leaves on each tree is small, which is also caused by the low dimension of the samples. In this case, similar samples are treated as the same sample, which would reduce the diversity of the samples in the dataset. Low diversity in the dataset increases the sensitivity of the algorithm. Therefore, asynch-SGBDT would be sensitive to the number of workers using the HIGGS dataset in our experimental settings.

The different sensibilities to the change in the number of workers between the HIGGS dataset and real-sim, Webspam, Criteo, KDD2012 datasets match conclusions 5 and 6 in section 5.3.5.

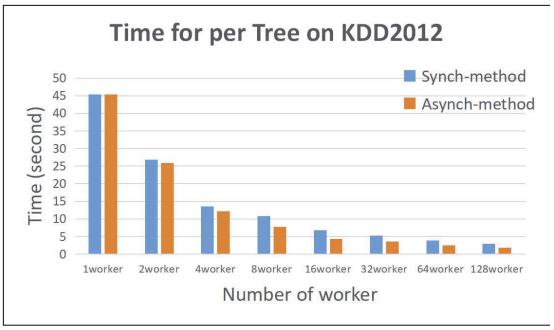
6.8.2 Corollary Experiment

It is easy to gain following corollary.

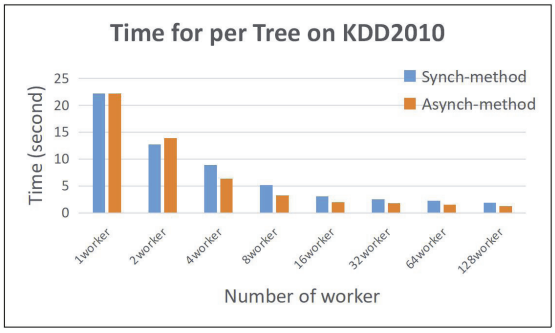
Corollary 1 Under following conditions that $1.n$ workers share the same sub-dataset with sampling rate is $sampling$ when $kn < iter < (k+1)n, k \in \mathbb{Z}$ which $iter$ is the number of current iterations, asynch-SGBDT uses learning rate v and $\tau = iter\%n$, the asynch-SGBDT process in n workers system is equal to the sequential sampling gradient step GBDT training process with learning rate being nv and sampling rate being $sampling$.



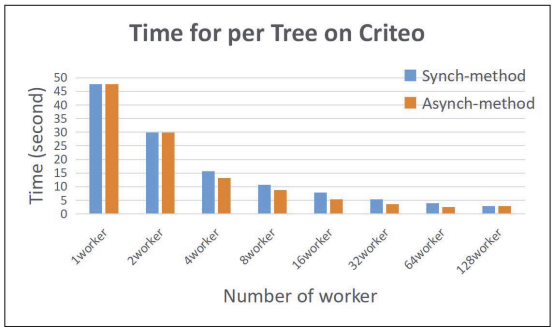
(a) Time for real-sim



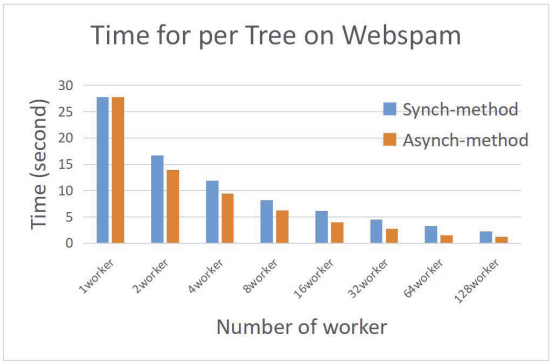
(b) Time for KDD2012



(c) Time for KDD2010



(d) Time for Criteo



(e) Speedup for webspam

Figure 7: Time comparison for asynchronous parallel GBDT (asynch-SGBDT) and synchronous parallel GBDT (LightGBM)

Based on the equivalent relation between asynch-SGDBT and sequential sampling gradient-step GBDT process and the conclusion that when sampling rate and learning rate is small enough, asynch-SGBDT do not reduce the convergence speed of GBDT training process and the do not reduce correct rate of final model, we can gain following predictable phenomenon that

Predictable Phenomenon In sampling GBDT training process, when learning rate v and sampling rate $sampling$ is small enough, the iteration of model reaching certain goal function is inversely proportional to the learning rate.

Above phenomenon shows that in sampling sequential GBDT training process, when v and $sampling$ is small enough, there is a definite quantitative relation between the number of iteration and learning rate. We test above phenomenon with the dataset KDD2010, learning rate is 0.0001 and sampling rate is 0.2.

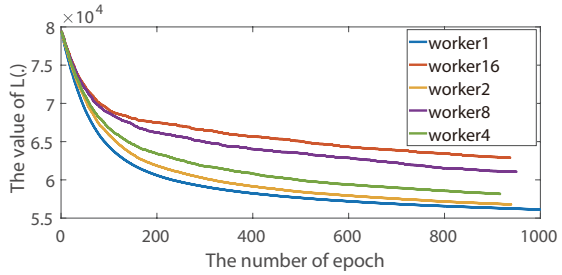
We offer this corollary experiment to make reader reproduce experiments without much code modification and show the equivalent relation between asynch-SGDBT and sequential sampling gradient-step GBDT process.

7 Conclusion

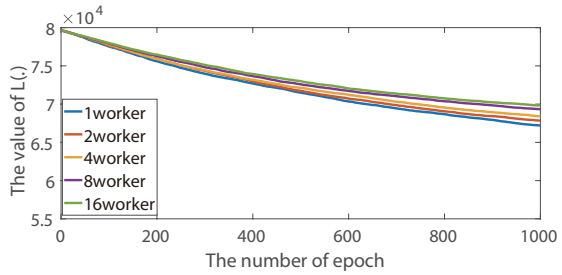
We propose a novel algorithm, asynch-SGBDT, that provides good compatibility for the parameter server. The theoretical analyses and experimental results show that a small iteration step, small sampling rate, large number of workers, and high-dimensional sparsity of the sample datasets lead to a fast convergence speed and high scalability. Specifically, asynch-SGBDT reaches a high speedup when asynch-SGBDT uses high-dimensional sparse datasets.

References

[1] M. Richardson, E. Dominowska, and R. J. Ragno, "Predicting clicks: estimating the click-through rate for new ads," pp. 521–530, 2007.
[2] "Adapting boosting for information retrieval measures," *Information Retrieval*, 2009.
[3] Y. Zhang and A. Haghani, "A gradient boosting method to improve travel time prediction," *Transportation Research Part C-emerging Technologies*, vol. 58, pp. 308–324, 2015.
[4] R. Caruana and A. Niculescumizil, "An empirical comparison of supervised learning algorithms," pp. 161–168, 2006.
[5] J. H. Friedman, "Greedy function approximation: A gradient boosting machine," *Annals of Statistics*, vol. 29, no. 5, pp. 1189–1232, 2001.



(a) sampling rate of 0.6



(b) sampling rate of $5 * 10^{-6}$

Figure 8: Sensitivity between the normal sampling rate and extremely small sampling rate on HIGGS dataset

[6] ——, "Stochastic gradient boosting," *Computational Statistics Data Analysis*, vol. 38, no. 4, pp. 367–378, 2002.

[7] P. Sun, T. Zhang, and J. Zhou, "A convergence rate analysis for logitboost, mart and their variant," in *International Conference on Machine Learning*, 2014, pp. 1251–1259.

[8] T. Chen and C. Guestrin, "Xgboost: A scalable tree boosting system," pp. 785–794, 2016.

[9] S. Chaturapruek, J. C. Duchi, and C. Ré, "Asynchronous stochastic convex optimization: the noise is in the noise and sgd don't care," in *Advances in Neural Information Processing Systems 28*, C. Cortes, N. D. Lawrence, D. D. Lee, M. Sugiyama, and R. Garnett, Eds. Curran Associates, Inc., 2015, pp. 1531–1539. [Online]. Available: http://papers.nips.cc/paper/

[10] A. V. Dorogush, V. Ershov, and A. Gulin, "Catboost: gradient boosting with categorical features support." *arXiv: Learning*, 2018.

[11] J. Jiang, B. Cui, C. Zhang, and F. Fu, "Dimboost: Boosting gradient boosting decision tree to higher dimensions," in *Proceedings of the 2018 International Conference on Management of Data*. ACM, 2018, pp. 1363–1376.

[12] L. M. Surhone, M. T. Tennoe, S. F. Henssonow, and L. Breiman, "Random forest," *Machine Learning*, vol. 45, no. 1, pp. 5–32, 2010.

[13] G. Biau, "Analysis of a random forests model," *Journal of Machine Learning Research*, vol. 13, no. 1, pp. 1063–1095, 2012.

[14] I. Mukherjee, C. Rudin, and R. E. Schapire, "The rate of convergence of adaboost," *Journal of Machine Learning Research*, vol. 14, no. 1, pp. 2315–2347, 2013.

[15] G. Ke, Q. Meng, T. Finley, T. Wang, W. Chen, W. Ma, Q. Ye, and T.-Y. Liu, "Lightgbm: A highly efficient gradient boosting decision tree," in *Advances in Neural Information Processing Systems 30*, I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, Eds. Curran Associates, Inc., 2017, pp. 3146–3154. [Online]. Available: http://papers.nips.cc/paper/6907-lightgbm-a-highly-efficient-gradient-boosting-decision-tree.pdf

[16] H. Zhang, S. Si, and C. Hsieh, "Gpu-acceleration for large-scale tree boosting," *arXiv: Machine Learning*, 2017.

[17] Q. Meng, G. Ke, T. Wang, W. Chen, Q. Ye, Z. Ma, and T. Liu, "A communication-efficient parallel algorithm for decision tree," pp. 1279–1287, 2016.

[18] S. Tyree, K. Q. Weinberger, K. Agrawal, and J. Paykin, "Parallel boosted regression trees for web search ranking," pp. 387–396, 2011.

[19] Y. Shi, J. Li, and Z. Li, "Gradient boosting with piece-wise linear regression trees," 2018.

[20] Z. Kalal, J. Matas, and K. Mikolajczyk, "Weighted sampling for large-scale boosting," in *British Machine Vision Conference 2008, Leeds, September*, 2008.

[21] R. Appel, T. Fuchs, and P. Perona, "Quickly boosting decision trees: pruning underachieving features early," in *International Conference on International Conference on Machine Learning*, 2013, pp. III–594.

[22] A. Gupta, R. Ravi, and A. Sinha, "Boosted sampling:approximation algorithms for stochastic optimization," in *ACM Symposium on Theory of Computing, Chicago, Il, Usa, June*, 2004, pp. 417–426.

[23] J. Ye, J. H. Chow, J. Chen, and Z. Zheng, "Stochastic gradient boosted distributed decision trees," in *Acm Conference on Information Knowledge Management*, 2009, pp. 2061–2064.

[24] J. Langford, A. J. Smola, and M. Zinkevich, "Slow learners are fast," in *International Conference on Neural Information Processing Systems*, 2009, pp. 2331–2339.

[25] F. Niu, B. Recht, C. Re, and S. J. Wright, "Hogwild!: A lock-free approach to parallelizing stochastic gradient descent," *Advances in Neural Information Processing Systems*, vol. 24, pp. 693–701, 2011.

[26] M. Li, D. G. Andersen, J. W. Park, A. J. Smola, A. Ahmed, V. Josifovski, J. Long, E. J. Shekita, and B. Su, "Scaling distributed machine learning with the parameter server," pp. 583–598, 2014.

[27] J. Jiang, J. Jiang, B. Cui, and C. Zhang, "Tencentboost: A gradient boosting tree system with parameter server," in *IEEE International Conference on Data Engineering*, 2017, pp. 281–284.

[28] T. Hastie, R. Tibshirani, J. H. Friedman, and J. Franklin, *The Elements of Statistical Learning*, 2015.

[29] J. C. Duchi, *Introductory Lectures on Stochastic Convex Optimization*. Park City Mathematics Institute, Graduate Summer School Lectures, 2016.

[30] T. Chen, M. Li, Y. Li, M. Lin, N. Wang, M. Wang, T. Xiao, B. Xu, C. Zhang, and Z. Zhang, "Mxnet: A flexible and efficient machine learning library for heterogeneous distributed systems," *Statistics*, 2015.

[31] M. Abadi, "Tensorflow: learning functions at scale," *Acm Sigplan Notices*, vol. 51, no. 9, pp. 1–1, 2016.

[32] E. P. Xing, Y. Yu, Q. Ho, W. Dai, J. K. Kim, J. Wei, S. Lee, X. Zheng, P. Xie, and A. Kumar, "Petuum," in *The ACM SIGKDD International Conference*, 2015, pp. 1335–1344.

APPENDIX

7.1 basic transformation

The Proof begin at iteration step

$$\mathbf{F}^{j+1} = \mathbf{F}^{j} - vA_{k(j)}P_{k(j)}L'_{base}(\mathbf{F}^{k(j)})$$

The minimum of $L(\mathbf{F})$ is $\mathbf{F}^*$. And Let $\beta = APL'_{base}(\mathbf{F})$. Then

$$\begin{aligned}
&(\mathbf{F}^{j+1} - \mathbf{F}^*) = (\mathbf{F}^{j} - \mathbf{F}^*) - vAPL'_{base}(\mathbf{F}^{k(j)}) \\
&+ v^2 L'(\mathbf{F}^{k(j)})^T P^T A^T APL'(\mathbf{F}^{k(j)}) \\
&+ v^2\beta^2_{k(j)} \\
&(\mathbf{F}^{j+1} - \mathbf{F}^*)^2 = (\mathbf{F}^{j} - \mathbf{F}^*)^2 + v^2\beta^2_{k(j)} \\
&- 2v((\mathbf{F}^{j} - \mathbf{F}^{k(j)})^T A^j PL'(\mathbf{F}^{j}) + (\mathbf{F}^{j} - \mathbf{F}^{k(j)})) \\
&- 2v((\mathbf{F}^{j} - \mathbf{F}^{k(j)})^T \beta_j + (\mathbf{F}^{j} - \mathbf{F}^{k(j)})^T(\beta_j - \beta_{k(j)}) \\
&+ (\mathbf{F}^{k(j)} - \mathbf{F}^*)^T \beta_{k(j)})
\end{aligned}$$

we will find the lower bound of

$$(\mathbf{F}^{j}-\mathbf{F}^{k(j)})^T\beta_j+(\mathbf{F}^{j}-\mathbf{F}^{k(j)})^T(\beta_j-\beta_{k(j)})+(\mathbf{F}^{k(j)}-\mathbf{F}^*)^T\beta_{k(j)}$$

This part is consist of first item

$$(\mathbf{F}^{j} - \mathbf{F}^{k(j)})^T\beta_j \tag{21}$$

The second item

$$(\mathbf{F}^{j} - \mathbf{F}^{k(j)})^T(\beta_j - \beta_{k(j)}) \tag{22}$$

And the thrid item

$$(\mathbf{F}^{k(j)} - \mathbf{F}^*)^T\beta_{k(j)} \tag{23}$$

7.2 Extra Lemma

Before continue proof, we need following extra lemma to support our next proof.

Lemma 1

$$\begin{aligned}
&\mathbb{E}(\mathbf{F}^{j} - \mathbf{F}^{i})APL'_{base}(\mathbf{F^j}) \geq \\
&\mathbb{E}(L(\mathbf{F}^{j}) - L(\mathbf{F}^{i}) + \frac{c}{2}\left\|\mathbf{F}^{j} - \mathbf{F}^{i}\right\|^2) - C_1\left\|\mathbf{F}^{j} - \mathbf{F}^{i}\right\|
\end{aligned}$$

Proof.

$$\begin{aligned}
&\mathbb{E}(\mathbf{F}^{j} - \mathbf{F}^{i})APL'_{base}(\mathbf{F^j}) \\
&= \mathbb{E}(\mathbf{F}^{j} - \mathbf{F}^{i})P_{fix}L'_{base}(\mathbf{F^j}) + \mathbb{E}(\mathbf{F}^{j} - \mathbf{F}^{i})(AP - P_{fix})L'_{base}(\mathbf{F^j})
\end{aligned}$$

For matrix multiplication is Linear operation. Thus,

$$\begin{aligned}
&\mathbb{E}(\mathbf{F}^{j} - \mathbf{F}^{i})APL'_{base}(\mathbf{F^j}) \\
=&\mathbb{E}(\mathbf{F}^{j} - \mathbf{F}^{i})P_{fix}L'_{base}(\mathbf{F^j}) \\
+&\mathbb{E}(\mathbf{F}^{j} - \mathbf{F}^{i})(\mathbb{E}AP - P_{fix})L'_{base}(\mathbf{F^j}) \\
\geq&\mathbb{E}(L(\mathbf{F}^{i}) - L(\mathbf{F}^{j}) + \frac{c}{2}\left\|\mathbf{F}^{j} - \mathbf{F}^{i}\right\|^2 \\
-&\left\|\mathbf{F}^{j} - \mathbf{F}^{i}\right\|\left\|(AP - P_{fix})L'_{base}(\mathbf{F^j})\right\|)
\end{aligned}$$

for any observation vector $\mathbf{Q}$, the vector $(AP - P_{fix})L'_{base}(\mathbf{F^j})$ always is the vector like

$$\begin{aligned}
&(AP - P_{fix})L'_{base}(\mathbf{F^j}) \\
&= (..., m_s\frac{\sum_{(x_r,y_r)\in leaf_J} m_r\ell'(y_r, F_r)}{\sum_{(x_r,y_r)\in leaf_s} m_r} - m_s\ell'(x_s, F_s), ...)^T
\end{aligned}$$

Thus, we can gain it norm upper bound that:

$$\begin{aligned}
&\left\|m_s\frac{\sum_{(x_r,y_r)\in leaf_J} m_r\ell'(y_r, F_r)}{\sum_{(x_r,y_r)\in leaf_s} m_r} - m_s\ell'(x_s, F_s)\right\| \\
=&m_s\left\|\frac{\sum_{(x_r,y_r)\in leaf_J} m_r(\ell'(y_r, F_r) - \ell(y_s, F_s))}{\sum_{(x_r,y_r)\in leaf_s} m_r}\right\| \\
\leq&\frac{m_s}{\sum_{(x_r,y_r)\in leaf_s} m_r}\left\|\sum_{(x_r,y_r)\in leaf_J}(m_r(\ell'(y_r, F_r) - \ell'(y_s, F_s)))\right\| \\
\leq&\frac{m_s}{\sum_{(x_r,y_r)\in leaf_s} m_r}\sum_{(x_r,y_r)\in leaf_J}\|\lambda m_r\|F_r - F_s\|\| \\
\leq&m_s\lambda\delta
\end{aligned}$$

Thus, $\left\|(AP - P_{fix})L'_{base}(\mathbf{F}^j)\right\| \leq \lambda\delta m_{max}\sqrt{\zeta}$, and $\mathbb{E}(\mathbf{F}^{j} - \mathbf{F}^{i})APL'_{base}(\mathbf{F^j}) \geq \mathbb{E}(L(\mathbf{F}^{j}) - L(\mathbf{F}^{i}) + \frac{c}{2}\left\|\mathbf{F}^{j} - \mathbf{F}^{i}\right\|^2 - \lambda\delta m_{max}\sqrt{\zeta}\left\|\mathbf{F}^{j} - \mathbf{F}^{i}\right\|)$ □

7.3 The bound of Eq. 21

We will show the lower bound of $(\mathbf{F}^{j} - \mathbf{F}^{k(j)})^T\beta_j$.

$$\begin{aligned}
&\mathbb{E}(\mathbf{F}^{j} - \mathbf{F}^{k(j)})^T\beta_j \\
\geq&\mathbb{E}(L(\mathbf{F}^{j}) - L(\mathbf{F}^{k(j)}) + \frac{c}{2}\left\|\mathbf{F}^{j} - \mathbf{F}^{k(j)}\right\|^2 - \lambda\delta\sqrt{\zeta}\left\|\mathbf{F}^{j} - \mathbf{F}^{k(j)}\right\|) \\
=&\mathbb{E}(\sum_{i=k(j)}^{j-1}(L(\mathbf{F}^{i+1}) - L((\mathbf{F})^{\mathbf{i}})) + \frac{c}{2}\left\|\mathbf{F}^{j} - \mathbf{F}^{k(j)}\right\|^2 - \\
&\lambda\delta m_{max}\sqrt{\zeta}\left\|\mathbf{F}^{j} - \mathbf{F}^{k(j)}\right\|)
\end{aligned}$$

Here, we have to show the bound of $\sum_{i=k(j)}^{j-1}(L(\mathbf{F}^{i+1}) - L((\mathbf{F})^{\mathbf{i}}))$

$$\begin{aligned}
&\mathbb{E}\sum_{i=k(j)}^{j-1}(L(\mathbf{F}^{i+1}) - L((\mathbf{F})^{\mathbf{i}})) \\
=&\mathbb{E}\sum_{i=k(j)}^{j-1}\sum_{\beta_j\text{'s } s\text{th component}\neq 0}\ell(y_s, F_s^{i+1}) - \ell(y_s, F_s^{i}) \\
\leq&\mathbb{E}\sum_{i=k(j)}^{j-1}(\mathbf{F}^{i+1} - \mathbf{F}^{i})^T A_i P_i L'(\mathbf{F}^{j}) \\
\leq&v\mathbb{E}\sum_{i=k(j)}^{j-1}L'(\mathbf{F}^{k(i)})^T P_{k(i)}^T A_{k(i)}^T A_i P_i L'(\mathbf{F}^{j}) \leq v\tau\rho M^2
\end{aligned}$$

Thus

$$\begin{aligned}
&\mathbb{E}(\mathbf{F}^{j} - \mathbf{F}^{k(j)})^T\beta_j \\
\geq&\mathbb{E}\frac{c}{2}\left\|\mathbf{F}^{j} - \mathbf{F}^{k(j)}\right\|^2 - v\tau\rho M^2 - \lambda\delta m_{max}\sqrt{\zeta}\left\|\mathbf{F}^{j} - \mathbf{F}^{k(j)}\right\|
\end{aligned}$$

7.4 The bound of Eq. 22

$$\begin{aligned}
&\mathbb{E}(\mathbf{F}^{j} - \mathbf{F}^{k(j)})^T(\beta_j - \beta_{k(j)}) = \\
&v\mathbb{E}\sum_{i=k(j)}^{j-1}A_i P_i L'(\mathbf{F}^{k(j)})(\beta_j - \beta_{k(j)}) \geq -2v\rho\tau M^2
\end{aligned}$$

7.5 The bound of Eq. 23

$$\begin{aligned}
&(\mathbf{F}^{k(j)} - \mathbf{F}^*)^T\beta_{k(j)} \geq \\
&\frac{c}{2}\left\|\mathbf{F}^{k(j)} - \mathbf{F}^*\right\|^2 - \lambda\delta m_{max}\sqrt{\zeta}\left\|\mathbf{F}^{k(j)} - \mathbf{F}^*\right\|
\end{aligned}$$

7.6 Algorithm Convergence Conclusions

$$
\begin{aligned}
&\mathbb{E}(\mathbf{F}^{j+1}-\mathbf{F}^*)^2\\
\leq&\mathbb{E}((\mathbf{F}^{j}-\mathbf{F}^*)^2+v^2M^2-2v(\frac{c}{2}\left\|\mathbf{F}^{k(j)}-\mathbf{F}^*\right\|^2\\
&-\delta\lambda m_{max}\sqrt{\zeta}\left\|\mathbf{F}^{k(j)}-\mathbf{F}^*\right\|-2v\rho\tau M^2\\
&+\frac{c}{2}\left\|\mathbf{F}^{j}-\mathbf{F}^{k(j)}\right\|^2-v\tau\lambda\rho M^2-\lambda\delta m_{max}\sqrt{\zeta}\left\|\mathbf{F}^{j}-\mathbf{F}^{k(j)}\right\|))\\
\leq&\mathbb{E}((\mathbf{F}^{j}-\mathbf{F}^*)^2-vc(\left\|\mathbf{F}^{j}-\mathbf{F}^{k(j)}\right\|^2+\left\|\mathbf{F}^{k(j)}-\mathbf{F}^*\right\|^2)\\
&+2v\delta\lambda m_{max}\sqrt{\zeta}(\left\|\mathbf{F}^{j}-\mathbf{F}^{k(j)}\right\|+\left\|\mathbf{F}^{k(j)}-\mathbf{F}^*\right\|)\\
&+2v^2M^2(3\rho\tau+\frac{1}{2}))
\end{aligned}
$$

For the item $\mathbb{E}\left\|\mathbf{F}^{j}-\mathbf{F}^{k(j)}\right\|^2+\mathbb{E}\left\|\mathbf{F}^{k(j)}-\mathbf{F}^*\right\|^2$, we can gain that

$$
\begin{aligned}
&\mathbb{E}\left\|\mathbf{F}^{j}-\mathbf{F}^{k(j)}\right\|^2+\mathbb{E}\left\|\mathbf{F}^{k(j)}-\mathbf{F}^*\right\|^2\\
=&\mathbb{E}(\left\|\mathbf{F}^{j}-\mathbf{F}^*\right\|^2-(\mathbf{F}^{j}-\mathbf{F}^{k(j)})^T(\mathbf{F}^{k(j)}-\mathbf{F}^*))\\
\geq&\mathbb{E}(\left\|\mathbf{F}^{j}-\mathbf{F}^*\right\|^2-\sum_{i=k(j)}^{j-1}(\mathbf{F}^{i+1}-\mathbf{F}^{i})^T(\mathbf{F}^{k(j)}-\mathbf{F}^*))\\
\geq&\mathbb{E}(\left\|\mathbf{F}^{j}-\mathbf{F}^*\right\|^2-v\sum_{i=k(j)}^{j-1}\left\|A_{k(i)}P_{k(i)}L'(\mathbf{F}^{k(i)})\right\|\left\|\mathbf{F}^{k(j)}-\mathbf{F}^*\right\|)\\
\geq&\mathbb{E}(\left\|\mathbf{F}^{j}-\mathbf{F}^*\right\|^2-v\sum_{i=k(j)}^{j-1}M\left\|\mathbf{F}^{k(j)}-\mathbf{F}^*\right\|)\\
\geq&\mathbb{E}(\left\|\mathbf{F}^{j}-\mathbf{F}^*\right\|^2-v\tau M\left\|\mathbf{F}^{k(j)}-\mathbf{F}^*\right\|)
\end{aligned}
$$

Thus,

$$
\begin{aligned}
&\mathbb{E}(\mathbf{F}^{j+1}-\mathbf{F}^*)^2\leq(1-vc)(\mathbb{E}(\mathbf{F}^{j}-\mathbf{F}^*))^2\\
&+2v\delta\lambda m_{max}\sqrt{\zeta}(\mathbb{E}\left\|\mathbf{F}^{j}-\mathbf{F}^{k(j)}\right\|)+(2v\delta\lambda m_{max}\sqrt{\zeta}\\
&+v^2c\tau M)(\mathbb{E}\left\|\mathbf{F}^{k(j)}-\mathbf{F}^*\right\|)\\
&+2v^2M^2(3\rho\tau+\frac{1}{2})\\
\leq&(1-vc)\mathbb{E}(\mathbf{F}^{j}-\mathbf{F}^*)^2+(4v\delta\lambda m_{max}\sqrt{\zeta}+cv^2\tau M)v\tau M\\
&+(2v\delta\lambda m_{max}\sqrt{\zeta}+cv^2\tau M)(\mathbb{E}\left\|\mathbf{F}^{j}-\mathbf{F}^*\right\|)\\
&+2v^2M^2(3\rho\tau+\frac{1}{2})\\
\leq&(1-vc)\mathbb{E}(\mathbf{F}^{j}-\mathbf{F}^*)^2\\
&+(2v\delta\lambda m_{max}\sqrt{\zeta}+cv^2\tau M)(\mathbb{E}\left\|\mathbf{F}^{j}-\mathbf{F}^*\right\|)\\
&+(4v\delta\lambda m_{max}\sqrt{\zeta}+cv^2\tau M)v\tau M+2v^2M^2(3\rho\tau+\frac{1}{2})
\end{aligned}
$$

We define C_1, C_2 as follows

$$
\begin{aligned}
C_1&=(2\delta\lambda m_{max}\sqrt{\zeta}+cv\tau M)\\
C_2&=(4\delta\lambda m_{max}\sqrt{\zeta}+cv\tau M)\tau M+2M^2(3\rho\tau+\frac{1}{2})
\end{aligned}
$$

Above in-equation is quadratic recurrence for $\mathbb{E}\left\|\mathbf{F}^{j}-\mathbf{F}^*\right\|^2$ and its convergence point is the fixed point, i.e.

$$
\begin{aligned}
&\mathbb{E}\left\|\mathbf{F}^{j+1}-\mathbf{F}^*\right\|^2-\mathbb{E}\left\|\mathbf{F}^{\infty}-\mathbf{F}^*\right\|^2\\
=&r\left(\mathbb{E}\left\|\mathbf{F}^{j}-\mathbf{F}^*\right\|^2-\mathbb{E}\left\|\mathbf{F}^{\infty}-\mathbf{F}^*\right\|^2\right)\\
&\mathbb{E}\left\|\mathbf{F}^{\infty}-\mathbf{F}^*\right\|^2=(\frac{C_1+\sqrt{C_1^2+4cvC_2}}{2c})^2 \qquad (24)\\
&r=1-vc(1-\frac{C_1}{C_1+\sqrt{C_1^2+4cvC_2}}) \qquad (25)
\end{aligned}
$$

To make our analysis clearly, we will make Eq.20 into following format and only care about the term structure.

$$
r=1-vc(1-\frac{1}{1+\sqrt{1+C_3(C_4+\frac{C_5}{(\tau+C_6)+\frac{C_7}{\tau+C_6}+C_8})}}) \qquad (26)
$$

Where C_3 to C_8 are positive and do not contain τ. In this form, we would know following result: when τ is large enough, with the increase of τ, the convergence speed would decrease, i.e. r would be increase.

To gain the r's sensitivity to τ, the standard method is to treat r as the function of M, τ, ρ and gain the $\nabla_\tau r(M,\rho,\tau)$. We expect that $\nabla_\tau r(M,\rho,\tau)$ is monotone to M and ρ. However, the form of $\nabla_\tau r(M,\rho,\tau)$ is complex and it is exhausting to gain $\nabla_\tau r(M,\rho,\tau)$.

However, we notice that the coefficients of τ is the term about M, i.e. about Ω and ρ. Above factor shows that when sampling rate is small, which leads to M and ρ are small, the influence of the changes of τ are small.

Thus, we also gain the result that small sampling rate, which leads to M and ρ are small, would reduce the convergence speed. This case is can be shown in Figure 8.

Because the coefficients of τ is also about ζ, we can also conduct the conclusion that the setting whose number of leaves entry is large would also reduce the influence of the changes of τ.

In Eq. 19, we can gain the result that Asynch-SGDBT would convergent to a range whose diameter is decided by the number of the tree, which decides δ. A large range usually leads to a better generalization effect.

When using the high instance diversity dataset and the GBDT setting GBDT trees contain massive leaves, sampling operation leads to small ρ and M, thus, high instance diversity dataset is apt to be accelerated.

Based on the result from Eq. 19 and Eq. 20, we can easily gain gerenal conclusions in section convergence analysis part.

7.7 Conclusions

Based on the result from Eq. 19 and Eq. 20, we can easily gain following result:

1. Asynch-SGBDT would converge to a range whose diameter is decided by the number of the tree. Usually We can gain a good generalization effect by using large diameter range.

2.Small sampling rate leads to slow convergence speed.

3.Small sampling rate increase the scalability.

4.Small learning rate increase the scalability.

5.Small learning rate leads to slow convergence speed.

6. Asynch-SGBDT is apt to accelerate the GBDT on a high-dimensional sparse dataset.

7. Asynch-SGBDT is apt to accelerate a GBDT whose trees contain massive leaves.