Parallel Gradient Boosting Decision Trees

Xuanang Chen (xc906@nyu.edu), Mengran Wang (mw1997@nyu.edu)

May 22, 2017

1 Introduction

Our project is to implement parallel gradient boosting decision trees using c++ and openmp. The parallelization can only be used in the tree building part because this is an step-wise algorithm. In our project, we parallel tree building phase with two method, by feature and by node, and the result has been analyzed and compared .

2 Gradient Boosting Decision Tree Algorithm

Gradient Boosting Decision Tree is a machine learning algorithm based on decision trees[1]. Decision tree method has many pros over those complicated algorithms. The derived decision model is easy to interpret and it can deal with outliers pretty well. However, it also has many disadvantages, such as its sensitivity to input and a high probability to overfit. So ensemble method comes to the rescue. They average a bunch of decision tree to reduce biases which not losing its ability to predict. It also reduce the probability of overfitting by combining a bunch of very simple decision trees which is very unlikely to learn some trivial patterns.Gradient boosting decision tree is one of ensemble method using decision trees.

For very machine learning algorithm, we have an objective function that we want to minimize. Here, we use

$$Obj = \sum_{i=1}^{n} l(y_i - \tilde{y_i}) + \sum_{k=1}^{K} \Omega(f_k)$$

In this function $\tilde{y_i}$ is the prediction value for x_i and y_i is the real value. So the first term measures the loss of the model. The second term is a regularization term used to improve the generalization ability of the model. In our project, we use square loss as our loss function and a regularization term is followed:

$$l(y_i, \tilde{y_i} = (y_i - \tilde{y_i})^2$$

$$\Omega(f_t) = \gamma T + \frac{1}{2}\lambda \sum_{j=1}^{T} w_j^2$$

where T is the number of leaves and w_j is the leave score. So our objective is to build a bunch of decision trees to minimize the objective function.

To learn the tree, additive learning is used which means start from the previous result, add a new tree (function) each step. Which is :

$$\tilde{y_i}^{(0)} = 0$$

$$\tilde{y_i}^{(1)} = f_1(x_i) = \tilde{y_i}^{(0)} + f_1(x_i)$$
$$\tilde{y_i}^{(2)} = f_1(x_i) + f_2(x_i) = \tilde{y_i}^{(1)} + f_2(x_i)$$

$$...$$

$$\tilde{y_i}^{(t)} = \tilde{y_i}^{(t-1)} + f_t(x_i)$$

Therefore, we can know that at round t, the objective function becomes:

$$Obj^t = \sum_{i=1}^{n} (2(\tilde{y_i}^{(t-1)} - y_i)f_t(x_i) + f_t(x_i)^2) + \Omega(f_t) + const$$

Here, we define

$$g_i = 2(\tilde{y_i}^{(t-1)} - y_i)$$

(first partial derivative)

$$h_i = 2$$

(second partial derivation of $(y_i - \tilde{y_i})^2$)

Given the funtion we want to optimize is a tree, so we can define the intance set in leaf j as $I_j = \{i|q(x_i) = j\}$ Therefore, the objective function becomes

$$obj^{(t)} \simeq \sum_{j=1}^{T} ((\sum_{i \subset I_j} g_i)w_j + \frac{1}{2}((\sum_{i \subset I_j} h_i + \lambda)w_j^2) + \gamma T$$

After build a tree, we can have g_i, h_i for each instance, so we can find the best tree struction of next step to minimize the objective function. However, there are almost infinite number of trees to construct, so we use greedy algorithm in practice. Basically, we grow the tree greedily: start from level 0, for each leaf node of the tree, we try to add a split, the change of objective after adding the split is:

$$Gain = \frac{1}{2}[\frac{G_l^2}{H_l + \lambda} + \frac{G_r^2}{H_R + \lambda} - \frac{(G_L + G_R)^2}{H_L + H_R + lambda}] - \gamma$$

To find the best split, we can just sort all the instance belong to this node according to each feature and use a left to right linear scan the find the max gain which will lead to the least added objective.

3 Parallel GBDT

In parallel part, we mainly use two parallel models[2]. First one is a straight forward model which parallels tree node in each level. Then we implemented a second parallel model which is more sophisticated but more reasonable in terms of our problems. We describe our two models below.

```
for each leaf node do
    for each feature do
        Sort the instances in the node by feature values;
        Linear scan to decide the best split;
    end
    Take the best split;
end
```

Algorithm 1: Parallel by node

Algorithm 1 is the first staright forward model that we think of for a decision tree framework.

```
Sort features globally first;
for each feature do
    for each leaf node do
        Linear scan to decide the best split;
    end
    Take the best split;
end
```

Algorithm 2: Parallel by feature

Then we alter the order of our for loop to parallel by each feature. In this model, since we sort our feature values globally first, we can skip the sort operation in each node. And the second model apparently needs more coding complexity, since we need to properly keep the status of each node's instances.

We test this on Courant Crunchy server which has 64 cores and 256G memory. Each sample has 40 attributes and tree depth is 6. Our test forest contains 10 trees.

4 Performance Analysis

For our two models, we compare the weak scaling, strong scaling and speedup performance. Our results are following.

We can see the second model does a much greater job, especially in multiple threads situations. And we find problems on our first straight forward model:

- It performs bad when encountering imbalance distributions of each node, which is quite common in decision tree tasks.
- In each node, it needs to sort local instances, which turns out to be quite time consuming when the number goes big.

5 Conclusion and Further Work

We have finished the whole training process and tree building part with pluralization. And timing the tree building process using Crunchy. We will integrate these two parts in the future. Tree Building part is done by Xuanang Chen and training phase is done by Mengran Wang.

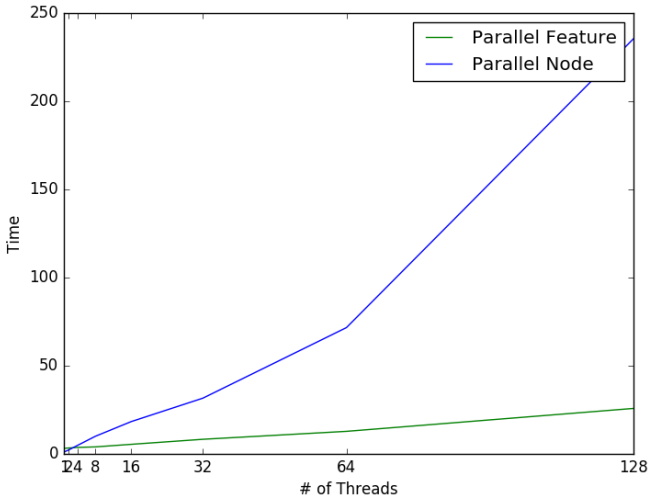

Figure 1: weak-scaling comparison

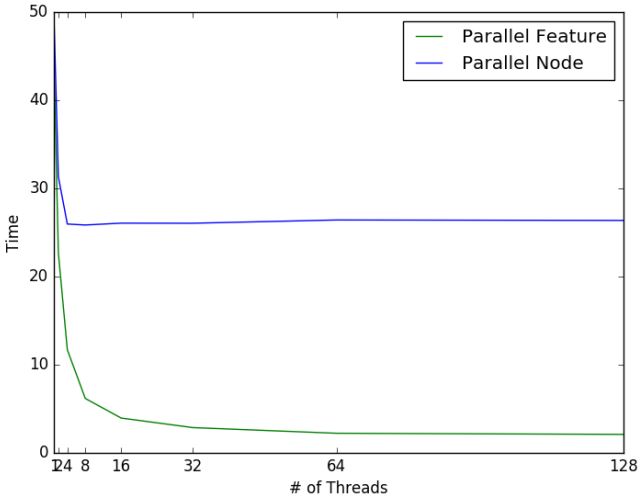

Figure 2: strong-scaling comparison

References

[1] Tianqi Chen. Introduction to boosted trees.

[2] Zhanpeng Fang. Parallel gradient boosting decision trees.

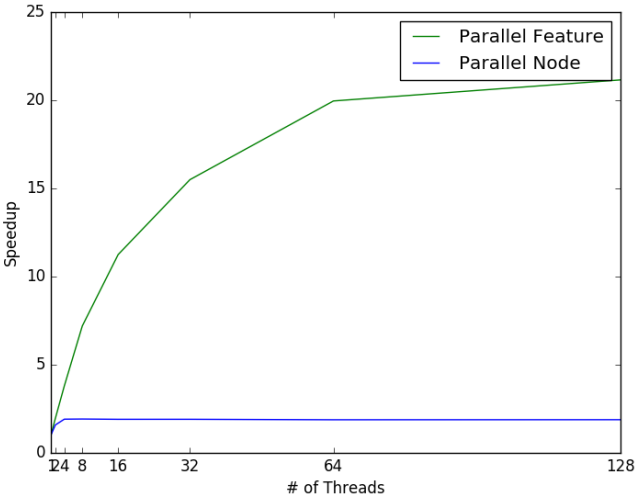

Figure 3: speed-up comparison