A New Data Classification Algorithm for Data-Intensive Computing Environments

Qizhi Deng, Longbo Zhang*, Xin Qian, Yali Chen, Fengying Wang
School of Computer Science
Shandong University of Technology
Zibo 255049, China
E-mail: roger-deng@163.com, zhanglb@sdut.edu.cn, qianzhaoxin163.com, ylchen870329@163.com, wfy@sdut.edu.cn
*Corresponding Author: zhanglb@sdut.edu.cn

Abstract—**In order to solve the problem of how to improve the scalability of data processing capabilities and the data availability which encountered by data mining techniques for Data-intensive computing, a new method of tree learning is presented in this paper. By introducing the MapReduce, the tree learning method based on SPRINT can obtain a well scalability when address large datasets. Moreover, we define the process of split point as a series of distributed computations, which is implemented with the MapReduce model respectively. And a new data structure called class distribution table is introduced to assist the calculation of histogram. Experiments and results analysis shows that the algorithm has strong processing capabilities of data mining for data-intensive computing environments.**

Keywords-MapReduce; Data-Intensive; SPRINT; Gini index

I. INTRODUCTION

Data-intensive computing applications become increasingly widespread; the data mining with large datasets for data-intensive computing tasks increasingly become a hot spot. Data-intensive computing tasks typically have the characteristics of large-scale in data storage (usually to TB or PB magnitude), dynamic changing, diversity (structure, semi-structure, unstructured, etc.), etc [1].These features bring many difficulties in data management operations across a cluster of commodity machines. With the wide and growing availability of MapReduce-capable compute infrastructures, it is necessary to ask whether such infrastructures could be use in parallelizing common data mining tasks such as tree learning. For many data mining operations, MapReduce may offer better scalability with vastly simplified deployment in a production setting.

MapReduce is a simple model for distributed computing which can eliminate many complexities such as data partitioning, handling machine failures, scheduling tasks across many machines, and performing inter-machine communication [2]. Despite the growing popularity of MapReduce [3], its application to certain standard data mining and machine learning tasks remains poorly understood. In the paper we only focus on the task of tree learning. Tree models are used in many applications because they are interpretable, can handle complex interactions, and can deal with both ordered and unordered features. At present, the data mining techniques for data-intensive computing environment is still infancy; the existing works mainly focus on building a high efficiency data mining model through taking advantage of scalability and fault tolerance of large-scale cluster system [4].

In this paper, we describe our experiences with developing and deploying a MapReduce based tree learner, an enhanced algorithm based on SPRINT decision tree algorithm. This algorithm use a new data structure called class distribution table, to reduce the complexities of the select of best split point and the partitions of attribute lists.

II. RELATED WORK

A. Distributed Decision Tree Classifier

Traditional distributed decision tree algorithm has many limitations due to the data all in memory and does not consider the actual problem of I/O and load balancing. Mehta [5] proposed a new decision tree algorithm SLIQ, with some extents; it overcomes the memory limitation using two data structures. Shafer et al. proposed the SPRINT algorithm on the basis of SLIQ and put forward the implementation on the distributed computing environment [6]. The SPRINT solved the problem of memory limitations and receives a better performance on time consuming and scalability than SLIQ. Caragea proposed a distributed decision tree algorithm-INDUS [7], on this basis; Dainan put forward a multi-distribution decision tree algorithm-DDTA (Distributed Decision Tree Algorithm) [8]. A MapReduce based tree learning model called PLANET was proposed by Panda [9], with some extends, it reduces the time consumption, but it also has the problem of memory limitations.

B. MapReduce Programming Model

This algorithm uses MapReduce [10, 11] to distribute and scale tree induction to large datasets. MapReduce can be described by two-phase distributed computation: map phase and reduce phase. Usually, large dataset partitioned into a set of disjoint units which are assigned to mapper workers. Each mapper scans through its assigned data and applies a user-specified map function to each record. The output of each mapper is a set of < key, value > pairs which are collected for Reduce phase. In reduce phase, the key-value pairs are grouped by key and are distributed to a series of reducer workers. Each reducer then applies a user-specified function to all the values for a key and outputs a final value for the key.

III. Classifier Of Data-Intensive Using MapReduce

Let $X=\{X_1, X_2,\ldots,X_N\}$ be a set of attributes with domains $D_{X_1}, D_{X_2},\ldots D_{X_N}$ respectively. Let Y be an output with domain D_Y. Consider a dataset $D = \{(x_i, y_i) \mid x_i \in D_{X_1} \times D_{X_2} \times \ldots D_{X_N}, y_i \in D_Y\}$ sampled from an unknown distribution, where the i^{th} data vector x_i has an output y_i associated with it. Given the dataset D, the goal in supervised learning is to learn a function (or model) $F: D_{X_1} \times D_{X_2} \times \ldots D_{X_N} \rightarrow D_Y$ that best approximates the true distribution of D. If D_Y is continuous, the learning problem is a regression problem; if D_Y is categorical, it is a classification problem [9].

A. Data Structure

A tree learner is built in two phases: a growth phase and a prune phase. In the growth phase, the tree is built recursively by partitioning the data until each partition is either "pure" (all members belong to the same class) or sufficiently small (a parameter set by the user) [10]. The form of the split used to partition the data depends on the type of the attribute used in the split. Splits for a continuous attribute A is with the form $value(A) < x$ where x is a value in the domain of A. Splits for a categorical attribute A are of the form $value(A) \in X$ where $X \subset domain(A)$. We consider only binary splits because they usually lead to more accurate trees. Some data structures are used in the algorithm, like follows.

Attribute List — We also keep attribute list for each attribute in the data. Entries in these attribute records, which maintained on disk, consist of an attribute value, a class label, and the index of the record from which these values were obtained. Initial lists for continuous attributes are sorted by attribute value once first created. The initial lists created from the training set are associated with the root of the classification tree. As the tree is grown and nodes are split to create new children, the attribute lists belonging to each node are partitioned and associated with the children. When a list is partitioned, the order of the records in the list is preserved; thus, partitioned lists never require resorting.

Block-Histogram—For this algorithm we also maintain two histograms. These histograms associated with each tree node corresponding with the local optimal split point of each data-block. Then choose the attribute be the best splits attribute which obtain the minimum gini value. These histograms also denoted as C_{above} and C_{below} are used to capture the class distribution of the attribute records at a given node. For categorical attributes also have a histogram associated with a node. Just like SPRINT, only one histogram is needed and it contains the class distribution for each value of the given attribute. We call this histogram a count matrix. Figure 1 shows the way to calculate of Block-Histogram.

Histogram—These histograms is different from SPRINT, it is a new data structure for building classification trees using MapReduce. Each data block maintain a record like <row, C_{above}, C_{below} >, the first row number of the data item in the data block; C_{below} maintains this distribution for attribute records that have already been processed, whereas C_{above} maintains it for those that have not. Each record represents a distribution of the class label in the data-block. The block histograms will obtain the initial value using these histograms which can simplify the complexity of the algorithm.

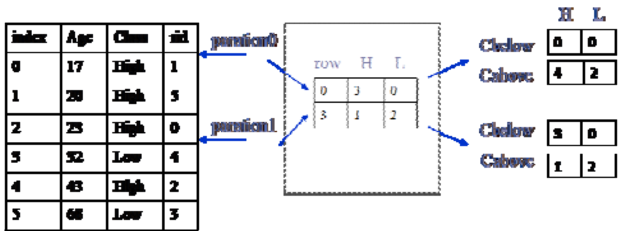

Figure 1. The calculation of Block-Histograms.

B. Description

At the heart of the algorithm is a controller method called BuildClassifier, a single machine that initiates, schedules and controls the entire tree induction process. In order to control and coordinate tree construction, the controller method constructs a tree using a series of MapReduce jobs, each of which builds different parts of the tree. At any point, a global TreeModel (TM) is maintained to contain the entire tree constructed so far.

As tree induction proceeds, the controller method add the root node into the node queue (NQ) and schedules MapReduce jobs to find split predicates at the nodes. Once a MapReduce job completes, the controller method updates TM with the nodes and their split predicates, and then updates NQ with new nodes at which split predicates can be partition. The main process of the algorithm is described as follows.

Algorithm 1 BuildClassifier

```
Require: NodeQueue NQ, TreeModel TM, Training record
 (x,y) ∈D, Attribute set Att
1.     T_root = new Node
2.     Initiate(T_root, D,Att)
3.     TM = T_root
4.     NQ.push_back(T_root)
5.     BuildTree(NQ)
```

The controller method creates a root node for the tree model at the beginning; then initiates it with the attribute lists, and marked with the root node. A global tree node queue NQ and tree model TM are maintained and initiated with T_{root}. Get the first decision-tree node marked T_{curr} from NQ, then build the sub-tree of T_{curr}. The details of build tree method described as follows.

Algorithm 2 BuidTree

```
Require: NodeQueue NQ, TreeModel TM, Training record
 (x,y) ∈D
1.     For All T_curr ∈NQ do
2.     If JudgeLeaf(T_curr) is false then
```

```
3.      bestSplit=FindBestSplit($T_{curr}$)
4.      $T_{curr}$→splitAtt=bestSplit→splitAtt
5.      If bestSplit→splitAtt is category then
6.          $T_{curr}$→leftAttSet=bestSplit→leftAttSet
7.          $T_{curr}$→rightAttSet=bestSplit→rightAttSet
8.      Else
9.          $T_{curr}$→splitValue=bestSplit→splitValue
10.     parationTrainingSet($T_{curr}$→D, leftD,rightD)
11.     remove($T_{curr}$→splitAtt)
12.     Create new node $T_{left}$, $T_{right}$
13.     Initiate($T_{left}$, leftD,Att)
14.     Initiate($T_{right}$,rightD,Att)
15.     $T_{curr}$→left=$T_{left}$
16.     $T_{curr}$→right=$T_{right}$
17.     NQ.push_back($T_{left}$)
18.     NQ.push_back($T_{right}$)
19.  Else
20.     $T_{curr}$→isLeaf = true
21.     $T_{curr}$→label =y //y is the most belonged label
```

In our tree mode, predicate evaluations at non-leaf nodes have only two outcomes, leading to binary splits. In the process of building tree, a judgment is needed at first, if the judgment is satisfied, the process will be ended and T_{curr} will be marked with the class table as a leaf node. If each partition of tree node T_{curr} is either "pure" (all members belong to the same class) or sufficiently small (a parameter set by the user). In Steps 3~10, the split point of each data nodes is found, the method FindBestSplit is the core operation of tree generation, we will discuss at next section. Using FindBestSplit, we will get the best split for T_{curr}. Then remove the split attribute from the attribute set and divide the attribute lists using the split value. Two tree nodes T_{left}, T_{right} are created and initiated with the divided attribute lists *leftD* and rightD, then pointed as the left node and right node of T_{curr} and put them into the node queue NQ. Until NQ is empty, the tree model is learned completely.

C. *Find Best Split*

Finding best split is very similar to the serial algorithm. In the serial version, processors scan the attribute lists either evaluating split points for continuous attributes or collecting distribution counts for categorical attributes. This does not change in the parallel algorithm, no extra work or communication is required while each processor is scanning its attribute-list partitions. We perform N split point parallel using Mappers, each Mapper independently processing 1/N of the training data. After that, use Reducer to choose the split which get the minimum split value.

We use the gini index, originally proposed in [3]. For a data set *D* containing examples from n classes, $Gini(D)$ is defined as $Gini(D) = 1 - \sum_{j=1}^{n} P_j^2$; where P_j is the relative frequency of class j in *D*. If a split divides *D* into two subsets D_1 and D_2, the index of the divided data $Gini_{split}(D)$ is given by $Gini_{split}(D) = {m_1}/{m}\, Gini(D_1) + {m_2}/{m}\, Gini(D_2)$. The advantage of this index is that its calculation requires only the distribution of the class values in each of the partitions. The method need to construct the Block-Histograms for each attribute and Histograms using a MapReduce job, and then find the best split for each attribute based on the calculation of gini index. The Mapper phase and Reducer phase are described as follows.

Algorithm 3 FindBestSplit::Mapper

```
Require:  Current node $T_{curr}$, Attribute set Att, Class set Y
1.    For all A ∈ Att do
2.        Class Count array countY for Y
3.        Index=firstreCord(Tcurr→D)
4.        For all (x,y)∈($T_{curr}$→D,Attribute list of A) do
5.            county[findY(Y,y)]++
6.        Output((findA(A)),( Index, countY))
```

Map Phase: Pseudocode used to describe the algorithms that are executed by each mapper appears in Algorithms 3. Each mapper collects the class distribution for each ordered attribute independently. In addition to the above outputs, each mapper maintains a table of key-value pairs. Keys are the index of attributes and values are the index of first record in each block of T_{curr}→D and the class distribution array.

Algorithm 4 FindBestSplit::Reducer

```
Require:  Key k, Value Set V, Attribute set Att, Class set Y
1.    For All k do
2.        If Att[k] is continues then
3.            For all distinct value ∈V do
4.                If sameBlock(value [i]) then
5.                    Output((k),( value [i], sumCount(value [i])))
```

Reduce phase: The reduce phase, which works on the outputs from the mappers, performs aggregations and computes the histograms for each available attribute. After that another MapReduce job is scheduled for the calculation of gini index, which using the output of this reduce phase. Finally, each reducer outputs the best split that it has been used for each tree node. The best split should include the split attribute, the left and right value set for category attribute and the split value for continues attribute.

IV. EXPERIMENTAL EVALUATIONS

Experiments based on Hadoop0.20.2 distributed platforms (nine PC machine), computer configured as CPU 2.93GHz, memory 2G, 150G hard drive, Ubuntu Linux system. Eclipse 3.3.2 as the development tool. One PC machine is the Namenode server, and the remaining eight are Datanode.

A. *Datasets for Experiments*

An often used benchmark in classification is UCI databases; however, its largest dataset contains only several million training examples. We choose KDD-Cup-1999 to be the training set which contains 4 million training examples and 40 attributes.

B. *Performance Analysis*

In order to analyze the performance of the algorithm, we design the experiments for the time efficiency of the algorithm, the parallel performance, accuracy, etc. As can be seen from Figure 2, when the data amount reaches a certain size, the response time will be apparent below than SPRINT.

The results show nice time-consuming performance. The drop in time-consuming is due to the MapReduce. When the data is larger enough, the data will be split into blocks with HDFS and processed as parallel. In contrast, the time-consuming of I/O is larger than the reduced time with parallel computing.

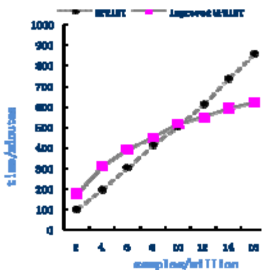

Figure 2. The comparison of time performance.

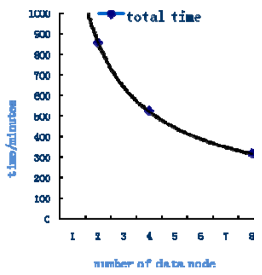

Figure 3. The time changing with data nodes number.

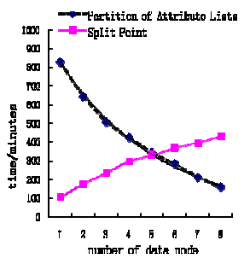

Figure 4. Time changing of finding split point and divided attribute list.

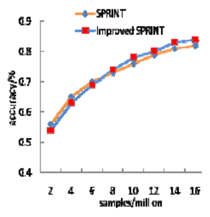

Figure 5. Accuracy trends.

Figure 3 shows that, in the case of same size of training set, the consuming time of the algorithm reduced significantly with the increasing of data nodes. Figures 4 presents the split point calculation time decrease linearly and the divided time of attribute lists increase with the number of data nodes increases. From Figures 5, we know that the accuracy of the algorithm is little change to SPRINT, because both of them using precise searching technique.

Through analysis and experimental results, we can know that the algorithm inherits the accuracy of SPRINT, gets a highly scalable and able to process large dataset. However, the division of the attribute lists is relatively complex, which limit the improvement of performance.

V. CONCLUSION

In this paper, we presented an enhanced algorithm for the data classification of data-intensive computing environment based on SPRINT and MapReduce. Combined with the features of MapReduce, a new data structure is introduced to assist finding the split point, which can decrease the I/O consuming and the complexity of algorithm. In some extent, the data availability problem is solved and it inherits the high scalability of SPRINT so that it has a strong ability to process large dataset. But the performance of the tree learner is not as well as continuous attribute when address category attribute, therefore, the next work we intends to improve the efficiency of algorithm through optimize the framework of the tree learner and the structure of attribute list.

ACKNOWLEDGMENT

The research is supported by the Natural Science Foundation of Shandong Province of China (Grant No. ZR2011FL013).

REFERENCES

[1] W. Peng, M. Dan, “Review of Programming Models for Data-Intensive Computing,” 11th ed., vol.47. Journal of Computer Research and Development, 2010, pp. 1993-2002.

[2] T. Richard, Kouzes, et al, “The Changing Paradigm of Data-Intensive Computing” 1th ed., vol.42, Computer, 2009, pp. 26-34.

[3] J. Dean, S. Ghemawat, Mapreduce: Simplified data processing on large clusters. In Symposium on Operating System Design and Implementation(OSDI), 2004.

[4] T. Ashish, S. Joydeep, et al, “Hive-A Warehousing Solution Over a Map-Reduce Framework,” PVLDB, Vol.2, no.2, 2009, pp. 1626-1629.

[5] M. Mehta, R. Agrawal and J. Rissanen, “SLIQ: A fast scalable classifier for data mining,” Lecture Notes in Computer Science, Vol.1057, Advances in Database Technology , 1996, pp. 18-32.

[6] J. Shafer, R. Agrawal, M. Mehta, “SPRINT: a Scalable Parallel Classifier for Data Mining,” //Proceedings of the 22nd VLDB Conference Mumbai(Bombay). Mumbai M organ Kaufmann, 1996, pp. 544-555.

[7] D. Caragea, A. Silvescu, “Decision tree induction from distributed heterogeneous autonomous data sources,” In Proc of the Conference on intelligent Systems Design and Applications. USA, 2003.

[8] D. Nan, J. Genlin, “Research and Implementation of ID3 Based on Distributed Database System,” Journal of Nanjing Normal University (Engineering and Technology), Vol.5, no.4, 2005, pp. 46-48.

[9] P. Biswanath, S. Joshua, et al, “PLANT: Massively Parallel Learning of Tree Ensembles with MapReduce,” VLDB Endowment, 2009, pp. 24-28.

[10] J. Dean, S. Ghemawat, “MapReduce: a flexible data processing tool. Commun,” CACM. Vol.1, no.53, 2010, pp. 72-77.

[11] J. Ekanayake, S. Pallickara, “MapReduce for Data Intensive Scientific Analysis,” Fourth IEEE International Conference on eScience, 2008, pp. 277-284