International Journal of Database Theory and Application
Vol.7, No.1 (2014), pp.49-60
http://dx.doi.org/10.14257/ijdta.2014.7.1.05

A MapReduce Implementation of C4.5 Decision Tree Algorithm

Wei Dai[1] and Wei Ji[*2]

School of Economics and Management, Hubei Polytechnic University, Huangshi 435003, Hubei, P.R.China

*[1]dweisky@163.com, [*2]jiweiit@163.com (Corresponding Author)*

Abstract

Recent years have witness the development of cloud computing and the big data era, which brings up challenges to traditional decision tree algorithms. First, as the size of dataset becomes extremely big, the process of building a decision tree can be quite time consuming. Second, because the data cannot fit in memory any more, some computation must be moved to the external storage and therefore increases the I/O cost. To this end, we propose to implement a typical decision tree algorithm, C4.5, using MapReduce programming model. Specifically, we transform the traditional algorithm into a series of Map and Reduce procedures. Besides, we design some data structures to minimize the communication cost. We also conduct extensive experiments on a massive dataset. The results indicate that our algorithm exhibits both time efficiency and scalability.

Keywords: *Decision tree, MapReduce, Hadoop*

1. Introduction

Decision trees are one of the most popular methods for classification in various data mining applications [1-2] and assist the process of decision making [3]. A decision tree is a directed tree with a root node which has no incoming edges and all other nodes with exactly one incoming edges, known as decision nodes. At the training stage, each internal node split the instance space into two or more parts with the objective of optimizing the performance of classifier. After that, every path from the root node to the leaf node forms a decision rule to determine which class a new instance belongs to.

One of the well-known decision tree algorithms is C4.5 [4-5], an extension of basic ID3 algorithm [6]. The improvements of C4.5 include: (1) employ information gain ratio instead od information gain as a measurement to select splitting attributes; (2) not only discrete attributes, but also continuous ones can be handled; (3) handling incomplete training data with missing values; (4) prune during the construction of trees to avoid over-fitting [7-8].

However, with the increasing development of cloud computing [9] as well as the big data challenge [10-12], traditional decision tree algorithms exhibit multiple limitations. First and foremost, building a decision tree can be very time consuming when the volume of dataset is extremely big, and new computing paradigm should be applied for clusters. Second, although parallel computing [13] in clusters can be leveraged in decision tree based classification algorithms [14-15], the strategy of data distribution should be optimized so that required data for building one node is localized and meanwhile the communication cost of minimized.

To this end, in this paper we propose a distributed implementation of C4.5 algorithm using MapReduce computing model, and deploy it on a Hadoop cluster. Our goal is to accelerate the construction of decision trees and also ensure the accuracy of classification by

ISSN: 2005-4270 IJDTA

leveraging parallel computing techniques. Specifically, our contributions can be summarized as follows:

We propose several data structures customized for distributed parallel computing environment;

We propose a MapReduce implementation of original C4.5 algorithm with a pipeline of Map and Reduce procedures;

We empirically prove the efficiency and scalability of our method with extensive experiments on a synthetical massive dataset.

The remains of this paper are organized as follows. Section 2 provides some preliminaries. Then our MapReduce implementation of C4.5 is proposed in Section 3. Empirical experiments are conducted in Section 4. Finally, the paper is concluded in Section 5.

2. Preliminaries

In this section, we will briefly introduce the background of decision trees and C4.5 algorithm, as well as MapReduce computing model.

2.1. Decision Trees and C4.5

A decision tree is a classifier which conducts recursive partition over the instance space. A typical decision tree is composed of internal nodes, edges and leaf nodes. Each *internal node* is called *decision node* representing a test on an attribute or a subset of attributes, and each edge is labeled with a specific value or range of value of the input attributes. In this way, internal nodes associated with their edges split the instance space into two or more partitions. Each *leaf node* is a terminal node of the tree with a *class label*. For example, Figure 1 provides an illustration of a basic decision tree, where circle means decision node and square means leaf node. In this example, we have three splitting attributes, *i.e.*, age, gender and criteria 3, along with two class labels, *i.e.*, YES and NO. Each path from the root node to leaf node forms a *classification rule*.

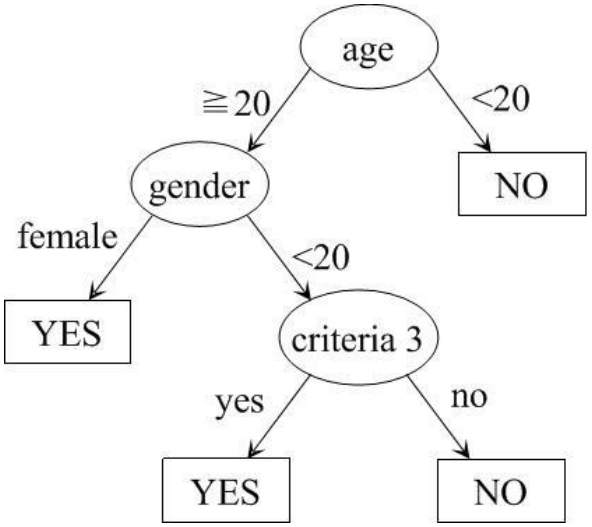

Figure 1. Illustration of Decision Tree

The general process of building a decision tree is as follows. Given a set of training data, apply a measurement function onto all attributes to find a best splitting attribute. Once the splitting attribute is determined, the instance space is partitioned into several parts. Within each partition, if all training instances belong to one single class, the algorithm terminates. Otherwise, the splitting process will be recursively performed until the whole partition is assigned to the same class. Once a decision tree is built, classification rules can be easily generated, which can be used for classification of new instances with unknown class labels.

C4.5 [4] is a standard algorithm for inducing classification rules in the form of decision tree. As an extension of ID3 [5], the default criteria of choosing splitting attributes in C4.5 is *information gain ratio*. Instead of using information gain as that in ID3, information gain ratio avoids the bias of selecting attributes with many values.

Algorithm 1 C4.5(T)

Input: training dataset T; attributes S.
Output: decision tree $Tree$.

if T is NULL **then**
  return failure
end if
if S is NULL **then**
  return $Tree$ as a single node with most frequent class label in T
end if
if $|S| = 1$ **then**
  return $Tree$ as a single node S
end if
set $Tree = \{\}$
for $a \in S$ **do**
  set $Info(a,T) = 0$, and $SplitInfo(a,T) = 0$
  compute $Entropy(a)$
  for $v \in values(a,T)$ **do**
    set $T_{a,v}$ as the subset of T with attribute $a = v$
    $Info(a,T) += \frac{|T_{a,v}|}{|T_a|} Entropy(a_v)$
    $SplitInfo(a,T) += -\frac{|T_{a,v}|}{|T_a|} log \frac{|T_{a,v}|}{|T_a|}$
  end for
  $Gain(a,T) = Entropy(a) - Info(a,T)$
  $GainRatio(a,T) = \frac{Gain(a,T)}{SplitInfo(a,T)}$
end for
set $a_{best} = \underset{a}{argmax}\{GainRatio(a,T)\}$
attach a_{best} into $Tree$
for $v \in values(a_{best},T)$ **do**
  call $C4.5(T_{a,v})$
end for
return $Tree$

Figure 2. C4.5 Algorithm Description

Let C denote the number of classes, and $p(S, j)$ is the proportion of instances in S that are assigned to j -th class. Therefore, the entropy of attribute S is calculated as:

$$Entropy(S) = -\sum_{j=1}^{C} p(S, j) \times \log p(S, j) \quad . \tag{1}$$

Accordingly, the information gain by a training dataset T is defined as:

$$Gain(S,T) = Entropy(S) - \sum_{v \in Values(T_s)} \frac{|T_{S,v}|}{|T_S|} Entropy(S_v) \quad , \tag{2}$$

where $Values(T_S)$ is the set of values of S in T, T_s is the subset of T induced by S, and $T_{S,v}$ is the subset of T in which attribute S has a value of v.

Therefore, the information gain ratio of attribute S is defined as:

$$GainRatio(S,T) = \frac{Gain(S,T)}{SplitInfo(S,T)} \quad , \tag{3}$$

where $SplitInfo(S,T)$ is calculated as:

$$SplitInfo(S,T) = -\sum_{v \in Values(T_S)} \frac{|T_{S,v}|}{|T_S|} \times \log \frac{|T_{S,v}|}{|T_S|} \quad . \tag{4}$$

The whole process of C4.5 algorithm is described in Algorithm 1. The information gain ratio criteria computation is performed in lines 11~21 using above equations, and a recursive function call is done in Line 25.

2.2. MapReduce

MapReduce programming model is used for parallel and distributed processing of large datasets on clusters [16]. There are two basic procedures in MapReduce: Map and Reduce. Typically, the input and output are both in the form of key/value pairs. As shown in Figure 2, after the input data is partitioned into splits with appropriate size, Map procedure takes a series of key/value pairs, and generates processed key/value pairs, which are passed to a particular reducer by certain partition function; Later, after data sorting and shuffling, the Reduce procedure iterates through the values that are associated with specific key and produces zero or more outputs.

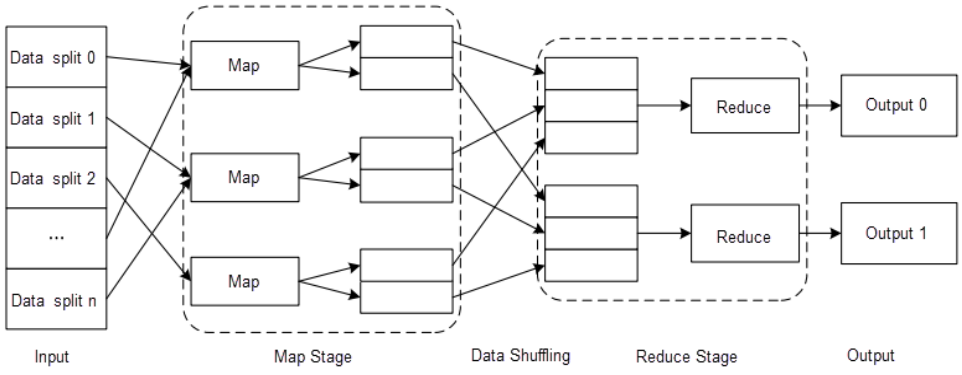

Figure 3. Description of MapReduce Programming Model

As an open source implementation of MapReduce, Hadoop [17] has two major components: HDFS (Hadoop Distributed File Systems) [18] and Hadoop MapReduce. The architecture is illustrated in Figure 4, where NameNode is the master node of HDFS handling metadata, and DataNode is slave node with data storage in terms of blocks. Similarly, the master node of Hadoop MapReduce is called JobTracker, which is in charge of managing and scheduling several tasks, and the slave node is called TaskTracker, where Map and Reduce procedures are actually performed. A typical deployment of Hadoop is to assign HDFS node and MapReduce node on the same physical computer for the consideration of localization and moving computation to data [19-20]. As you will see in Section 4, we apply this deployment in our experiments.

3. Proposed Algorithm

In this section, we present our proposed MapReduce implementation of C4.5 algorithm. We first introduce several data structures, and then present the MapReduce implementation of C4.5 in the form of a pipeline of Map and Reduce procedures.

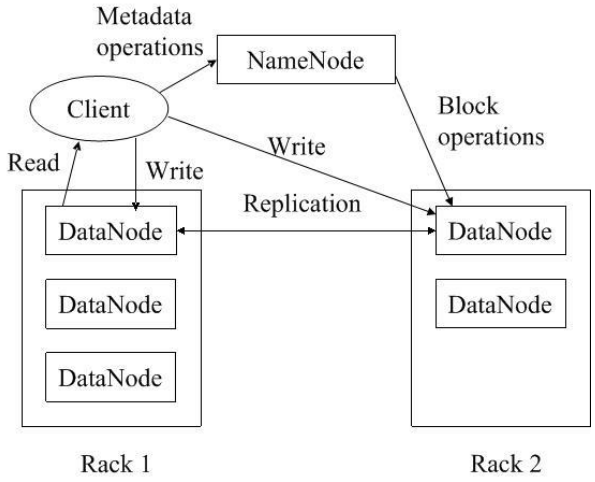

Figure 4. Architecture of HDFS

3.1. Data Structure

In a big data environment, the dataset is quite massive, and the data required for building decision trees is typically dynamic. For example, generating a node needs not only the information of itself but also data from other nodes, and therefore introduce communication cost between nodes. Besides, when the dataset cannot fit in memory, reading data from external storage also increases the I/O cost. To this end, designing appropriate data structures for our distributed parallel algorithm is certainly necessary. A fundamental assumption is that the memory is not enough to fit the whole dataset.

The first data structure is *attribute table*, which stores the basic information of attribute a, including the row identifier of instance row_id, values of attribute $values(a)$ and class labels of instances c.

The second one is *count table*, which computes the count of instances with specific class labels if split by attribute a. That is, two fields are included: class label c and a count cnt.

The last one is *hash table*, which stores the link information between tree nodes

$node\_id$ and row_id, as well as the link between parent node $node\_id$ and its branches $subnode\_id_1, subnode\_id_2, \ldots$.

Note that on a distributed parallel environment, dataset vertically partitioned to maximally preserve localization characteristic for the sake of efficiency. In our implementation, we split the dataset by equally assign attributes to several nodes. Suppose we have *M* attributes and *N* nodes, and therefore there are [*M/N*] on first *N*−1 nodes, and the remaining attributes are stored on the last node.

3.2 MapReduce Implementation

Now we discuss how to implement C4.5 algorithm using above three data structures. Generally, the whole process is composed of four steps: data preparation, selection, update and tree growing.

Algorithm 2 Data Preparation

1: **procedure** MAP_ATTRIBUTE(row_id, $(a_1, a_2, \ldots, a_M, c)$)
2: emit $(a_j, (row\_id, c))$
3: **end procedure**
4: **procedure** REDUCE_ATTRIBUTE($(a_j, (row\_id, c))$)
5: emit $(a_j, (c, cnt))$
6: **end procedure**

Figure 5. Description of Data Preparation

3.2.1. Data Preparation: Before executing the algorithm, the first thing we need is to convert the traditional relational table based data into above three data structure for further MapReduce processing. As shown in algorithm 2, procedure MAP_ATTRIBUTE transforms the instance record into *attribute table* with attribute $a_j (j = 1,2,...M)$ as key, and row_id and class label c as values. Then, REDUCE_ATTRIBUTE computes the number of instances with specific class labels if split by attribute a_j, which forms the *count table*. Note that *hash table* is set to null at the beginning of process.

Algorithm 3 Attribute Selection

1: **procedure** REDUCE_POPULATION($(a_j, (c, cnt))$)
2: emit (a_j, all)
3: **end procedure**
4: **procedure** MAP_COMPUTATION($(a_j, (c, cnt, all))$)
5: compute $Entropy(a_j)$
6: compute $Info(a_j) = \frac{cnt}{all} Entropy(a_j)$
7: compute $SplitInfo(a_j) = -\frac{cnt}{all} log \frac{cnt}{all}$
8: emit $(a_j, (Info(a_j), SplitInfo(a_j))$
9: **end procedure**
10: **procedure** REDUCE_COMPUTATION($(a_j, (Info(a_j), SplitInfo(a_j)))$)
11: emit $(a_j, GainRatio(a_j)$
12: **end procedure**

Figure 6. Description of Attribute Selection

3.2.2. Selection: After we have attribute table and count table, the first step is to select best attribute a_{best}. As shown in Algorithm 3, it has one Map function and two Reduce functions. First, the REDUCE_POPULATION procedure takes the number of instances for each attribute/value pair to aggregate the total size of records for given attribute a_j. Next, after MAP_COMPUTATION procedure computes the information and split information of a_j, procedure REDUCE_COMPUTATION computes the information gain ratio, just as described in Lines 11~21 in Algorithm 1. Last, a_j with the maximum value of $GainRatio(a_j)$ will be selected as splitting attribute a_{best}.

Algorithm 4 Update Tables

1: **procedure** MAP_UPDATE_COUNT($(a_{best}, (row\_id, c))$)
2: emit $(a_{best}, (c, cnt'))$
3: **end procedure**
4: **procedure** MAP_HASH((a_{best}, row_id))
5: compute $node\_id = hash(a_{best})$
6: emit $(row\_id, node\_id)$
7: **end procedure**

Figure 7. Description of Update Tables

3.2.3. Update: Now we have to update count table and hash table. As shown in Algorithm 4, Procedure MAP_UPDATE_COUNT reads a record from attribute table with key value equals to a_{best}, and emits the count of class labels. Procedure MAP_HASH assigns $node\_id$ based on a hash value of a_{best} to make sure that records with same values are split into the same partition.

3.2.4. Tree Growing: Since we generate nodes in Algorithm 4, now we need to grow the decision tree by building linkage between nodes, as shown in Algorithm 5. For the next iteration, compute $node\_id$ as shown in Line 2. If the value remains the same, it means that $node\_id$ is a leaf node, as shown in Lines 3~5. Otherwise, a new sub node is attached in Lines 6~7.

Algorithm 5 Tree Growing

1: **procedure** MAP((a_{best}, row_id))
2: compute $node\_id = hash(a_{best})$
3: **if** $node\_id$ is same with the old value **then**
4: emit $(row\_id, node\_id)$
5: **end if**
6: add a new subnode
7: emit $(row\_id, node\_id, subnode\_id)$
8: **end procedure**

Figure 8. Description of Tree Growing

The whole process is a MapReduce pipeline with a sequential combination of Map and Reduce procedures described above. Among them, only the data preparation is a one time task, and the remaining are repetitive. The terminal condition is all $node\_id$ become leaf nodes, and then a decision tree is built.

4. Experiments

In this section, we provide experiments to evaluate the performance of our MapReduce implementation of C4.5 algorithm.

We have a Hadoop cluster deployed on 4 PCs with 2.11 GHz dual-core CPU, 1G RAM and 200G hard disk. We use each core as a Hadoop node, and thus we have 8 nodes. On each physical core, both a HDFS and MapReduce nodes are deployed. We let one of them as HDFS *NameNode* and MapReduce *JobTracker* (*i.e.*, master), and the remaining nodes act as HDFS *DataNode* and MapReduce *TaskTracker* (*i.e.*, slave).

Since the efficiency of C4.5 is theoretically and empirically proved, in our study we are concerned with the time efficiency of parallel version of C4.5 in big data environment. Given the fact that there lacks of a massive dataset for classification, we use a synthetic data collection. In our dataset, there are 6 nine attributes, which is described in table 1, and 2 class labels A and B. The number of training instances in our experiment varies from 0.5 to 3 millions. Note that: (1) we have $\lfloor 6/4 \rfloor = 1$ on first three PCs, and the last one stores 3 attributes; (2) on each dual-core PC, memory is shared as in [21].

Table 1. Attributes Description in Dataset

Attribute	Description
salary	Values between 1,000 and 150,000
age	Valued between 20 and 80
gender	Male or female
loan	Values between 0 and 500,000
commission	If salary ≦ 60,000, commission is 0; otherwise, commission = 2%*salary
marital status	Single, married or divorced

4.1. Performance on Single Node

In this subsection, we compare the performance of MapReduce implementation with the original C4.5 on single node. Figure 9 illustrates the results, from which we have the following observations. First, the larger the dataset is, the more time consuming it is to build the decision tree. Second, the execution time of our MapReduce based algorithm is much less than the original C4.5 algorithm as the size of dataset increases. Therefore, it is proved that our proposed method outperforms the sequential version even on a single node environment.

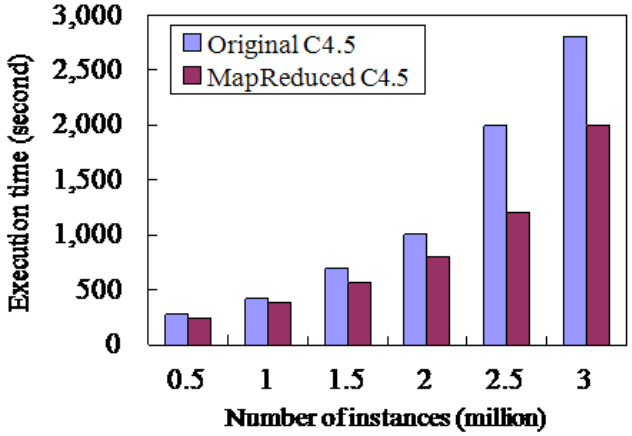

Figure 9. Performance on Single Node with Various Numbers of Instances

4.2. Scalability

Now we evaluate the performance of proposed MapReduce based C4.5 algorithm in a distributed parallel environment. The scalability evaluation includes two aspects: (1) performance with different numbers of nodes, and (2) performance with different size of training datasets. As mentioned earlier, we have 8 nodes totally on 4 physical computers, and our training dataset varies from 0.5 to 3 millions in terms of the number of instances.

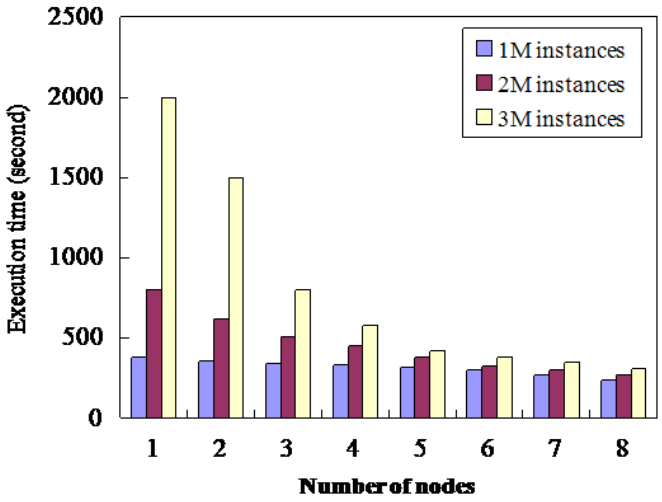

Figure 10. Performance on Different Numbers of Nodes with Specific Sample

First, we test the scalability performance on different numbers of nodes given specific training dataset. Figure 10 illustrates the execution time of our MapReduce based C4.5 algorithm with different numbers of nodes when the number of instances is 1, 2 and 3 millions respectively. We can observe that the overall execution time decreases when the number of nodes increases. This indicates that the more nodes are involved for computing, the more efficient the algorithm will be.

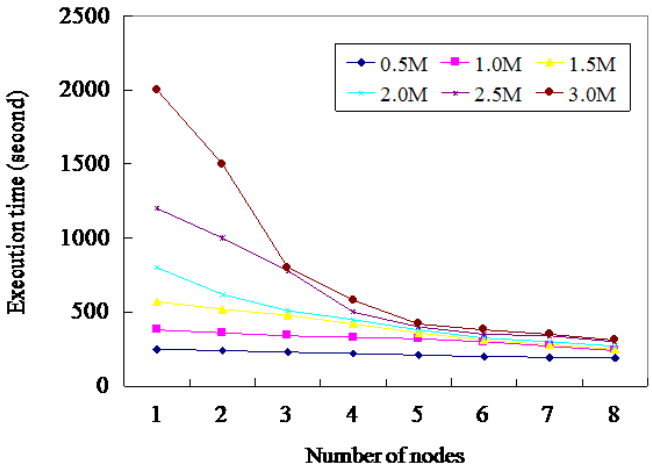

Figure 11. Performance for Different Sample Size on Different Numbers of Nodes

On the other hand, to evaluate the scalability with various sizes of training data, we also conduct experiment on different sample datasets. Figure 11 shows the execution time of different size of sample datasets, where the legend denotes the numbers of instances in training data. Moreover, Figure 12 provides the speedup performance of various numbers of training instances as the number of nodes increases, where speedup is a popular measurement of parallel algorithm defined as the ratio of execution time of sequential algorithm to that of the parallel algorithm with specific numbers of processors.

From Figures 11 and 12, we can see that: (1) the larger the training dataset we use, the more cost of execution time; (2) the more nodes we use, the less of execution time; (3) if enough nodes are leveraged, even the size of dataset is big, the performance can be close to the optimal one. For example, given 3.0 millions of training data, if we use 8 nodes, the execution time is close to that of the smallest dataset, *i.e.*, 0.5 millions. That is to say, by leveraging more nodes, we can solve big data problem just like the old times.

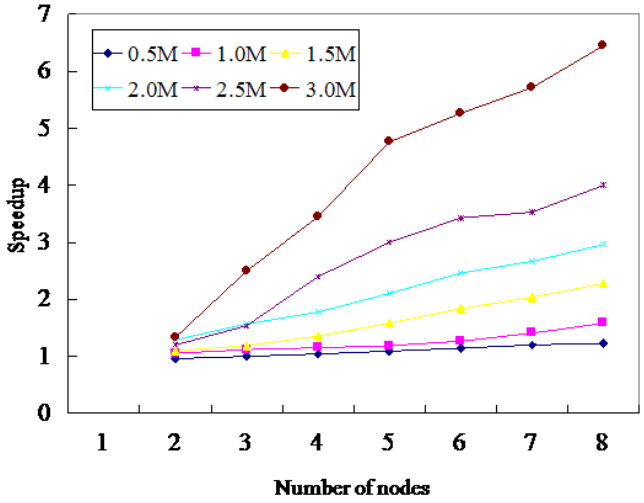

Figure 12. Speedup for Different Sample Size on Different Numbers of Nodes

5. Conclusions

In this paper, we propose a MapReduce implementation of C4.5 algorithm. The motivation is that with the increasingly development of cloud computing and big data, traditional sequential decision tree algorithms cannot fit any more. For example, as the size of training data grows, the process of building decision trees can be very time consuming. Besides, with the volume of dataset increases, the algorithm has a high cost on I/O operation because the the required data cannot fit in memory. To solve above challenges, we therefore propose a parallel version of C4.5 based on MapReduce. In order to evaluate the efficiency of our method, we also conduct extensive experiments on a synthetic massive dataset. The empirical results indicate that our MapReduce implementation of C4.5 algorithm exhibit both time efficiency and scalability.

In future works, we might want to further investigate other typical data mining and machine learning algorithms using MapReduce.

Acknowledgements

This study has been financially supported by Humanities and Social Science Youth Fund Project of Ministry of Education (No.13YJCZH028).

References

[1] H. I. Witten and E. Frank, "Data Mining: Practical machine learning tools and techniques", Morgan Kaufmann, **(2005)**.
[2] M. J. Berry and G. S. Linoff, "Data mining techniques: For marketing, sales, and customer support", John Wiley & Sons, Inc., **(1997**).
[3] J. R. Quinlan, "Decision trees and decision-making", IEEE Transactions on Systems, Man and Cybernetics, vol. 20, no. 2, **(1990)**, pp. 339-346.
[4] J. R. Quinlan, "C4.5: programs for machine learning", Morgan Kaufmann, **(1993)**.
[5] J. R. Quinlan, "Improved use of continuous attributes in C4.5", arXiv preprint cs/9603103, **(1996).**
[6] J. R. Quinlan, "Induction of decision trees", Machine Learning, vol. 1, no. 1, **(1986)**, pp. 81-106.
[7] D. Ventura and T. R. Martinez, "An empirical comparison of discretization methods", Proceedings of the Tenth International Symposium on Computer and Information Sciences, **(1995)**, pp. 443-450.
[8] H. Li and X. M. Hu, "Analysis and Comparison between ID3 Algorithm and C4. 5 Algorithm in Decision Tree", Water Resources and Power, vol. 26, no. 2, **(2008)**, pp. 129-132.
[9] M. Armbrust, A. Fox, R. Griffith, A. D. Joseph, R. Katz, A. Konwinski, G. Lee, D. Patterson, A. Rabkin, I. Stoica and M. Zaharia, "A view of cloud computing", Communications of the ACM, vol. 53, no. 4, **(2010)**, pp. 50-58.
[10] D. Howe, M. Costanzo, P. Fey, T. Gojobori, L. Hannick, W. Hide, D.P. Hill, R. Kania, M. Schaeffer, S.S. Pierre, S. Twigger, O. White and S.Y. Rhee. "Big data: The future of biocuration", Nature, vol.455, no.7209, **(2008)**, pp.47-50.
[11] S. LaValle, E. Lesser, R. Shockley, M. S. Hopkins and N. Kruschwitz, "Big data, analytics and the path from insights to value", MIT Sloan Management Review, vol. 52, no. 2, **(2011)**, pp. 21-31.
[12] P. Zikopoulos and C. Eaton, "Understanding big data: Analytics for enterprise class hadoop and streaming data", McGraw-Hill Osborne Media, **(2011)**.
[13] V. Kumar, A. Grama, A. Gupta and G. Karypis, "Introduction to parallel computing", Redwood City: Benjamin/Cummings, vol. 110, **(1994)**.
[14] K. W. Bowyer, L. O. Hall, T. Moore, N. Chawla and W. P. Kegelmeyer, "A parallel decision tree builder for mining very large visualization datasets", IEEE International Conference on Systems, Man, and Cybernetics, vol. 3, **(2000)**, pp. 1888-1893.
[15] J. Shafer, R. Agrawal and M. Mehta, "SPRINT: A scalable parallel classifier for data mining", Proc. 1996 Int. Conf. Very Large Data Bases, **(1996)**.
[16] J. Dean and S. Ghemawat, "MapReduce: simplified data processing on large clusters", Communications of the ACM, vol. 51, no. 1, **(2008)**, pp. 107-113.
[17] T. White, "Hadoop: the definitive guide", O'Reilly, **(2012)**.
[18] F. Wang, J. Qiu, J. Yang, B. Dong, X. Li and Y. Li, "Hadoop high availability through metadata replication",

Proceedings of the first international workshop on cloud data management, ACM, **(2009)**, pp. 37-44.
[19] C. A. Hansen, "Optimizing Hadoop for the cluster", **(2012)**.
[20] C. Zhang, H. D. Sterck, A. Aboulnaga, H. Djambazian and R. Sladek, "Case study of scientific data processing on a cloud using hadoop", High Performance Computing Systems and Applications, Springer Berlin Heidelberg, vol. 5976, **(2010)**, pp. 400-415.
[21] C. T. Chu, S. K. Kim, Y. A. Lin, Y. Y. Yu, G. Bradski, Y. N. Andrew and K. Olukotun, "Map-reduce for machine learning on multicore", NIPS, MIT Press, **(2006)**, pp. 281-288.

Authors

Wei Dai. He received his M.S.E. in Computer Science and Technology (2006) and PhD in Computer Application Technology (2012) from Wuhan University of Technology. Now he is full researcher of Economics and Management Department, Hubei Polytechnic University. His current research interests include different aspects of Intelligence Computing and Information Systems.

Wei Ji. He received his B.Ec. in International Economics and Trade (2003) from Wuhan University of Technology and MBA in Business Administration (2010) from Huazhong University of Science and Technology. Now he is full researcher of informatics at Economics and Management Department, Hubei Polytechnic University. His current research interests include different aspects of Financial Engineering and Information Computing.

