Research of Decision Tree on YARN Using MapReduce and Spark

Hua Wang1, Bin Wu1, Shuai Yang1, Bai Wang1, and Yang Liu1
1 School of Computer Science Beijing University of Posts and Telecommunications
Beijing, China

Abstract - Decision tree is one of the most widely used classification methods. For massive data processing, MapReduce is a good choice. Whereas, MapReduce is not suitable for iterative algorithms. The programming model of Spark is proposed as a memory-based framework that is fit for iterative algorithms and interactive data mining. In this paper, C4.5 is implemented on both MapReduce and Spark. The result of each layer of the decision tree can be kept in memory in the implementation on Spark. Through the experiments of C4.5, we observed an improvement of 950% on Spark than on MapReduce when the dataset is small. When the number of lines reached 50 million, Spark still kept an improvement of 73%. We concluded the algorithms and applications applicable for MapReduce and Spark. In the discussion section further experiments were performed to confirm our conclusions.

Keywords: MapReduce, Spark, RDDs, iterative algorithms, decision tree

1 Introduction

In recent years the size of data and information is presenting an explosive growth trend. With the restriction of the amount of memory and computing capability of traditional standalone mode, it is more and more difficult for traditional data mining tools to deal with TB level and PB level data. As a solution to deal with huge amount of data, parallel mechanism has attracted more and more attention. MPI, PVM and MapReduce [1] were all widely used in the past years.

Comparing with traditional parallel methods, MapReduce performs especially well when the size of datasets is large, and is relatively easy to use. By providing parallelization, fault tolerance, data distribution and load balancing in a transparent and easy-to-use way, MapReduce is widely accepted and used. With the implementation of MapReduce, Apache Hadoop is widely used. Hadoop is mainly composed of two parts: MapReduce and HDFS (Hadoop Distributed File System).

With the science development, a number of applications which are based on iterative algorithms [2] appear. Hadoop MapReduce [3] is based on an acyclic data flow model. With the output of the previous MapReduce job as the input of the next MapReduce job, the iterative programs can be accomplished. In such design, the data used in each iteration is reread and reprocessed, wasting a lot of time in I/O operation. Spark [4] is an open source project developed by UC Berkeley AMPLab. With the realization of RDDs [5], a distributed memory abstraction that lets programmers perform in-memory computations on large clusters, Spark provides RDDs transforms and actions for the users to use Spark easily. YARN [6] is the resource and applications manager of a cluster and supports the existence of multiple frameworks.

Decision tree learning is a powerful method for pattern classification. Most current researchers on decision tree mining focus on improving the mining algorithm which only improves the efficiency of the algorithm rather than the capability of the data to be processed. When the amount of data to be processed increases exponentially, it becomes unsuitable in the single point data mining platform. There are also some researches of decision tree on Hadoop. While the iterative algorithms such as decision tree and k-means are not suitable for the disk based frameworks like Hadoop. The memory based frameworks like Spark are proposed with a view to the shortness of MapReduce.

In this paper, we firstly got a thorough understanding of the mechanism of MapReduce and Spark. We found that the implement of RDDs makes Spark suitable for iterative algorithms. By parallelizing the phase of choosing the best split attribute, we implemented C4.5 on MapReduce. In the implementation of C4.5 on Spark, the intermediate result of each iteration is persisted in memory. In the experiments we got the time of each iteration of different sizes of data sets. We found that the implement of C4.5 performs better than that of MapReduce with an improvement of 73%-950%. We got the conclusions that Spark is suitable for iterative algorithms, which are I/O intensive, low computing density and use specific data sets. Considering the mechanism of Spark and the processing procedure, K-means was chosen to perform further experiments. K-means on Spark was about 33 times faster than that of MapReduce. When the lines of data set reached 150 million, Spark still kept an improvement of 400%. Related works are discussed in section 6, and in section 7 we summarized our conclusions and future work. We represented our acknowledgement in section 9.

2 Background

2.1 MapReduce
The MapReduce programming model consists of two functions, map and reduce. The process of MapReduce job is shown in Fig. 1.

![Fig. 1. The process of MapReduce job](image)

As illustrated in Fig. 1, the input data is divided into fixed size of splits (64M by default) by the MapReduce framework. A series of key/value pairs are generated from each split. Then each set of key/value pairs are assigned to a Map task which is designed by the user to implement specific logic, and a new set of intermediate key/value pairs are generated after the Map function. In the Reduce function, each reduce task consumes the (key, list<value>) tuples from map assigned to it. In this phase, a user defined function is invoked that transforms the (key, list<value>) to an output key/value pair. The framework distributes the reduce tasks across the cluster of nodes and deals with the transportation of the appropriate fragment of intermediate data to each reduce task.

As above, the output of Map is directly written into local disk after the shuffle phase. If the algorithm is iterative, the algorithm will read data from external stable storage systems at the start of each iteration. This wastes a lot of time in network bandwidth data replication, and disk I/O.

### 2.2 Spark

Spark is a distributed computing framework which is designed for low-latency and iterative computation on historical data. Spark provides an easy-to-program interface that is available in Java, Python, and Scala. The major facilities provided by Spark are as follows:

#### 2.2.1 Resilient Distributed Datasets (RDDs)

Spark provides a fault tolerant and efficient memory abstraction called Resilient Distributed Databases (RDDs). When a RDD is created, the users can decide which intermediate RDDs are to be kept in memory and control their partitioning to optimize data placement to get high-efficiency result. RDDs also provide fault tolerance by logging the transformations (map, reduceByKey, filter, etc.).

#### 2.2.2 The operations on RDDs

The operations on RDDs are mainly classified into two categories: transformations and actions. With the operations of transformations, the user can create a new dataset from an existing RDD. All transformations in Spark are lazy in case of that they do not compute their results right away. After the operation of actions, a value is returned to the driver program.

#### 2.2.3 Job Scheduling

When a job is committed to the master of the cluster, a DAG is built from the RDD’s lineage graph. A DAG consists of several stages. The stages are divided into two categories: shuffle map stage and result stage. Shuffle map stages are those that their results are input for another stage, while result stages are those that their tasks directly compute the action that initiated a job (count, collect, save, etc.).

#### 2.2.4 Shared Variables

Two common usage patterns of shared variables are provided by Spark: broadcast variables and accumulators. We can broadcast read-only variables and implement counters by using shared variables.

### 2.3 YARN (Hadoop 2.0)

YARN is the next generation of MapReduce. The programming model and data process engine in MRv1 are reused in MRv2. The principal change of MRv2 to MRv1 is that it split up the two major functionalities of JobTracker into separate daemons. The architecture of YARN with MapReduce and Spark as the applications is shown in Fig. 2.

![Fig. 2. Example of how Spark computes job stages.](image)

Totally speaking, YARN is also a Master/Slave architecture. ResourceManger is responsible for the uniform resource management and the schedule. When an application is submitted, an ApplicationMaster is needed to track and supervise the job.

### 3 Decision Tree

Decision Tree is one of the key Data Mining technologies and categorizations. In a Decision Tree, every internal node means a test on an attribute, every branch means the output of a test, and every leaf node store a class label. ID3 [7] is firstly developed by J. Ross Quinlan in 1986. C4.5 [8] is developed by J. Ross Quinlan in 1993, since then ID3 and C4.5 have been widely used and also have a lot development. In this paper, the
parallelization of C4.5 is put forward and realized by MapReduce and Spark. In the experiment of C4.5 with MapReduce and Spark, some conclusions are reached.

On account of the measure of information gain in ID3 is partial to the attributes that have a lot of lines in the data set, C4.5 chooses gain ratio as the extension of information gain. In this paper, C4.5 is selected to be parallelized and realized on MapReduce and Spark. C4.5 adopts the top-down and recursive method to construct a decision tree from the training items and the categories they belong to. The detail procedures are shown as below.

1) Get the input data set of DSet. Each item in DSet has some attribute values and a class label;
2) Find the gain ratio from splitting on each attribute att;
3) Let att_best be the attribute with the highest gain ratio;
4) Create a decision node that splits on att_best;
5) After splitting on att_best, some subcubes are formed. For each cube of CubeChild the subcubes, go back to 2) to get att_best1 of CubeChild. Att_best1 will be the child of the node formed in 4).

Additionally, some operations of pruning will be performed to overcome the excessive fitting.

The entropy of a data set to be classified is measured as:

$$info(D) = \sum_{i=1}^{m} p_i log_2 (p_i)$$  (1)

The $p_i$ means the probability of one item belongs to class $C_i$, and is measured by $|C_i|/|D|$. $info(D)$ is called the entropy of $D$. The information except $info(D)$ we need to get accurate classification of the data set is measured as:

$$info_A(D) = \sum_{j=1}^{v} \frac{|p_j|}{|D|} \times info(D)$$  (2)

The $\frac{|p_j|}{|D|}$ acts as the weight of the jth partition. $info_A(D)$ is the expected information according A to classify the items in D. The information gain is defined as the difference between the original information $info(D)$ and the new information $info_A(D)$:

$$Gain(A) = info(D) - info_A(D)$$  (3)

ID3 uses $Gain(A)$ to get the split attribute. While C4.5 uses split information to normalize information gain:

$$Split:info_A(D) = - \sum_{j=1}^{v} \frac{|p_j|}{|D|} \times log_2 \left( \frac{|p_j|}{|D|} \right)$$  (4)

The standard C4.5 used to split a node is gain ratio, which is shown as follows:

$$GainRatio(A) = \frac{Gain(A)}{SplitInfo(A)}$$  (5)

4 C4.5 on MapReduce and Spark

In order to do data mining on YARN using MapReduce and Spark, some tools and infrastructure are required. The architecture of the data mining system is shown in Fig. 3.

![Fig. 3. The architecture of the data mining system](image)

In this study, we used a cluster with ten nodes of Linux operating system as the base infrastructure of the whole system. On top of the infrastructure, we used HDFS (Hadoop Distributed File System) for data storage. Hadoop provides shell operations and APIs for the users to have access to the data stored on HDFS. On top of HDFS, YARN is chosen as the resource manager and applications master to manage the cluster. For data processing and analysis, both MapReduce and Spark are selected with the purpose of comparing the characteristics of MapReduce with Spark. With the MapReduce and Spark frames, it’s possible for us to develop parallel algorithms or application. Here Decision Tree is chosen as an example to be implemented on MapReduce and Spark.

4.1 C4.5 on MapReduce

The traditional decision tree algorithm is memory resident, which means that all the data sets are kept in memory during the whole formation process of the decision tree. In this case, the scalability of the algorithm is under restrictions. In this article, we discussed the parallelization of C4.5.

Through the analysis of the process of C4.5, we concluded that the most important part of C4.5 is the phase of the measurement of attribute selection. Choosing the best split attribute occupies most time of the decision tree generating phase. So it is the breakout of parallelizing C4.5 tree to get the greatest degree of this phase’s parallelization. In sequence of
the relative independence among different attributes, it is possible for us to use MapReduce to compute the related information needed to calculate the gain ratio of each attribute. Then, the main procedure can get the gain ratio rapidly and get the best split attribute. The main idea of parallelizing C4.5 tree likes the WordCount procedure to some degree. In this paper, we used breadth-first algorithm to get the result tree.

Map phase: Assuming that the training set is \( Node_0 \), there are \( m \) nodes in one layer of the tree. The nodes supposed to be satisfied with:

\[
Node_0 - Node' = Node_1 \cup Node_2 \cup \ldots \cup Node_m
\]  
(5)

The \( Node' \) is the set of the items that are in leaves.

The duty of map phase is to get the \(<key, value>\) form of the item in \( Node_0 \), and output the data as \( Node_1, Node_2, Node_3, \ldots, Node_m \). Key is the id of the \( Nodei \), the attribute \( att \), the value of \( att \), and the class value. The value is set 1. Map also has the duty to get the total line number of training set and the line numbers of \( Nodei \). These statistical works can be done in a single map task.

The reduce phase is to get the sum number of values that has the same key from the output of map phase. Then the \(<key, sum>\)s are output to HDFS. A combiner which is similar to reducer, is added before the reducer in order to reduce the size of the data to be transmitted through network. With the result of reduce output, it’s a simple job for us to get the gain ratio of each attribute in \( Nodei \) and get the split attribute that has the max gain ratio. The flow diagram of the process is shown in Fig. 4.

In the function of map, we can get the split of each line to get \(<id+att+value+class, 1>\), \(<id, 1>\), \(<"total", 1>\) as the output of map. The reducer gets the output of map so as to get the sum of the values that have the same key. The output of reducer is put on HDFS. With the information needed to get gain ratio, we can get the best attribute among the attributes that have not been the split nodes.

4.1 C4.5 on Spark

The C4.5 on Spark has the same parallel idea with C4.5 on MapReduce. As Spark has different APIs and operation from MapReduce, being familiar with Spark and its operations is necessary for us to write a Spark application. The diagram of the working flow of Spark is shown in Fig. 5.

An application is also called a driver on Spark. When a job is submitted to YARN, a runtime environment is created. At the same time, a service called BlockManager, which adopted an architecture of master/slaved, is started on each node. Then the application is transformed to a DAG. The DAGScheduler is on duty of executing every stage of the process. The C4.5 code based on Spark is shown in Fig. 6.
Early-stage preparations:
1. Spark and YARN configuration
2. Putting data on HDFS

Run C4.5Tree Class (the Driver Program)
SparkContext:
The constructor: new SparkContext(master, appName, [SparkHome], [jars]) is called to initialize SparkContext.
Initialization:
Read and initialize attributes and their possible values from meta file.
RDD:
The input training set is regarded as a RDD on Spark through textFile(path, minSplits): RDD[String].
flatMap:
Get a list through each input line, including:
1. <id,att>value<class, 1>
2. <id, 1>
3. <"total", 1>
Id means the unique number of a node on current layer.
reduceByKey:
Get the sum of the same key from the RDDs from flagMap.
generateTree:
Get the attribute that has the highest gain ratio in each node on current layer.

Fig. 6. The working process of C4.5 on Spark from flatMap. The reduceByKey works as the function of reducer in MapReduce to get the sum needed to get the gain ratio of each node. When the information required is worked out, it is possible for us to get the attribute that has the highest gain ratio in each node on current layer.

5 Experiments

Some experiments are conducted to evaluate the performance of our implementation. In this paper, we used a cluster with one master and 9 slaves. All these nodes have an internal memory of 4GB and 4 cores. Each node is installed with Red Hat 4.4.7-3.

The dataset of Lymphography Domain was used in our experiment on MapReduce and Spark. In order to get different size of datasets to evaluate the performance of MapReduce and Spark, the copy method is used to get assigned number of lines.

The Lymphography Domain Data Set was obtained from the University Medical Centre, Institute of Oncology, Ljubljana, Yugoslavia. It was provided by M. Zwitter and M. Soklic. There are 19 attributes including the class attribute. All attribute values in the database have been entered as numeric values. The number of lines of the databases used in this experiment is: 50 thousand, 500 thousand, 2 million, 5 million, 8 million, 10 million, 15 million, 30 million and 50 million. The size of a 50 thousand dataset is about 5M.

For that there are 6 layers of the decision tree, we record the time at the end of each layer’s iteration. The performance of C4.5 on MapReduce is shown in Fig. 7. In the following figures w means ten thousand.

![Fig. 7. The performance of C4.5 on MapReduce](image)

In this experiment, we found that the running time is close to the maximum in the 4th layer, for the reason that the layer 4th has the most nodes. During the generation phase of decision tree, matching the candidate rules which contains only the current node’s ancestor nodes takes most of the time of the whole phase. At the beginning, there are a small number of candidate nodes results in the short running time. With the number of candidate rules growing, the running time progressively grows. After the 4th layer is built, the running time of single layer reduces for the reducing of nodes. The performance of C4.5 on Spark is shown in Fig. 8.

![Fig. 8. The performance of C4.5 on Spark](image)

The performance curvilinear trend of C4.5 on Spark is almost the same as that of C4.5 on MapReduce except the beginning of the process. With the time used in reading data stored on HDFS and storing the dataset in memory, the running time of the first layer is relatively long. While after the first iteration, the running time of each iteration reduces. The trend after this goes almost the same as that of MapReduce. The running time reaches the peak at the 4th layer. The comparison of C4.5 between MapReduce and Spark is shown in Fig. 9.

From Fig. 9 we can find that when the number of lines is relatively small, i.e. 50 thousand, the running speed of Spark is much higher than that of MapReduce, at about 10.5 times difference. As the amount of data increases, the advantage of Spark reduces gradually. But the speed of C4.5 on Spark is still
higher than that of MapReduce, with about 1.73 times faster at a data set of 50 million lines.

6 Discussion

From the experiment in section V, we found that Spark is faster than MapReduce to a certain extent. Considering the characteristics of MapReduce, Spark and the executing process of C4.5, we can get the following conclusions:

1) MapReduce is not suitable for the processing of a small amount of data due to the starting time of a MapReduce job. Compared with MapReduce, Spark does not have this drawback. Even the size of data is very small, the job of Spark also runs fast.

2) With the ability to keep data in memory, Spark is especially fit for iterative algorithms. Spark has the ability of permitting a user to cache the data that will be reused in the algorithm. This is very flexible and useful. Spark saves the time in I/O of reading and writing intermediate result, which occupies a large part of the process of MapReduce.

3) Spark is fit for the situation repeatedly using specific dataset, which can be kept in memory. If the dataset always changes during the whole process, the advantage of Spark over MapReduce becomes relatively poor.

4) Spark is fit for I/O intensive applications. Extremely speaking, the size of dataset is large, but what we do is just to get the number of lines for n times. Spark is very suitable for this situation. While, if the computing density is very high, which takes more time than that of I/O, Spark’s advantage over MapReduce is not so obvious.

Through discussing the result of our experiment, it is concluded that Spark is specially fit for the algorithms that are I/O intensive and repeatedly use specific dataset. Among these, K-means [9] is a typical sample. K-means is a clustering algorithm, which aims to divide n items into k cluster, where the items in the same cluster are similar to each other, while the items in different clusters have low similarity. In the process of K-means, the input dataset, which can be kept in memory, will never change during the whole K-means process.

Besides, the logic of each iteration of K-means is simpler than that of C.5. The data of K-means in this paper are produced by a specific program. The data has 30 dimensions. It is about 5.37M of 10 thousand nodes. The test datasets are in numbers of 50 thousand, 50 thousand, 1 million, 2 million, 5 million, 10 million, 20 million, 50 million, 80 million and 150 million. The comparison of K-means between MapReduce and Spark is shown in Fig.10 and Fig. 11.

7 Related work

Nowadays, there are some studies about data mining based on Hadoop. Mahout [10] is an open source project which contains the implementation of common machine learning algorithms based on Hadoop. Oryx [11] is the open source machine learning project of Cloudera based on Hadoop. There are also some researches about data mining on Spark. For example, Spark mllib [12] is a Spark implementation of some common machine learning functionality, which contains binary classification, regression, clustering, etc..
data hub [13] is a big data platform based on Hadoop 2.0 and Spark, which also integrates Mahout and R statistics engine.

As to decision tree, [14] [15] and [16] provide some improvement strategies. There are also some researches about decision tree based on MapReduce. [17] and [18] are studies on the implementation of decision tree on MapReduce. Mahout also has the implementation of decision forest based on MapReduce. The research about decision tree on Spark is still rare, and there are also few studies on the comparison of advantages and applicable algorithms between MapReduce and Spark. In this paper we implemented C4.5 on both MapReduce and Spark, and concluded the situations suitable for Spark.

8 Conclusions and future work

As the use of Spark is becoming more and more widespread and YARN has become the new generation of Hadoop, the data mining based on YARN using both MapReduce and Spark has become a future trend. In this study, we implemented C4.5 on MapReduce and Spark. Through the analysis of the mechanism of MapReduce and Spark, it is found that Spark is suitable for I/O intensive and low computing density algorithms. When each iteration uses a specific dataset, Spark performs much better. Otherwise, Spark performs relatively poor. Further experiments of K-means is conducted to prove our conclusions.

This is a basic study where we parallelize C4.5 on MapReduce and Spark. We will try to implement more complicated algorithms to research how to take full advantage of Spark. Through our research of Spark, we will try to improve the performance of data mining algorithms. We will also integrate the algorithms on Spark to common data mining platforms.

9 Acknowledgement

This work is supported by the National Key Basic Research and Department (973) Program of China (No.2013CB329603) and the National Science Foundation of China (Nos:61373058, and 71231002). This work is also supported by the Special Cooperation Project of Beijing Municipal Commission of Education.

10 References


