Improvement of TF-IDF Algorithm Based on Hadoop Framework

Bin Li
Department of Computer Science
Colleague of Information Science & Technology
Jinan University
Guangzhou, China
gz-jnu-bg@hotmail.com

Yuan Guoyong
Department of Computer Science
Colleague of Information Science & Technology
Jinan University
Guangzhou, China
gdzsygy@gmail.com

***Abstract*-TF-IDF algorithm is often used in search engine, text similarity computation, web data mining, etc. These applications are often faced with the massive data processing. Therefore, how to calculate the tf-idf quickly and efficiently is very important. In this paper, we give a tf-idf algorithm based on the hadoop framework. Experiments show that in the case of massive data computing, the new method applying hadoop framework is more efficient than the traditional methods.**

Keywords-Hadoop, TF-IDF, distributed computing

I. INTRODUCTION

The tf-idf weight[1] (term frequency–inverse document frequency) is a weight often used in information retrieval and text mining. This weight is a statistical measure used to evaluate how important a word is to a document in a collection or corpus. The importance increases proportionally to the number of times a word appears in the document but is offset by the frequency of the word in the corpus.Tf-idf algorithm is often used in search engine, web data mining, text similarity computation and other applications. These applications are often faced with the massive data processing. So, how to calculate the tf-idf quickly and efficiently is very important.

II. TF-IDF ALGORITHM

A. The term count in the given document is simply the number of times a given term appears in that document. For the term t_i within the particular document d_j,its term frequency is defined as follows:

$$\mathrm{tf}_{i,j} = \frac{n_{i,j}}{\sum_k n_{k,j}}$$

In the formula ,ni, j is the number of occurrences of the considered term (t_i) in document d_j, and the denominator is the sum of number of occurrences of all terms in document d_j.

B. The inverse document frequency is a measure of the general importance of the term The formula are defined as follows:

$$\mathrm{idf}_i = \log \frac{|D|}{|\{j : t_i \in d_j\}|}$$

In the formula ,| D | is the total number of documents in the corpus; $|\{j : t_i \in d_j\}|$ is the number of documents where the term ti appears (that is $n_{i,j} \neq 0$).

C. The tf-idf weight of term is the product of tf and idf. The formula are defined as follows[2]:

$$(\mathrm{tf\text{-}idf})_{i,j} = \mathrm{tf}_{i,j} \times \mathrm{idf}_i$$

III. ABOUT HADOOP FRAMEWORK

Hadoop is an open source distributed parallel programming framework which can run on a large number of cluster. HDFS distributed File System and MapReduce computation model is its two main components. HDFS is highly fault-tolerant and is designed to be deployed on low-cost hardware. It provides high throughput access to application data and is suitable for applications that have large data sets[3,5]. MapReduce is a programming model and an associated implementation for processing and generating large data sets[4]. Users specify a map function that processes a <key, value> pair to generate a set of intermediate <key, value> pairs, and a reduce function that merges all intermediate values associated with the same intermediate key[5,6]. The relationship between HDFS and MapReduce is shown as follow:

IV. THE IMPROVEMENT AND IMPLEMENTATION OF TF-IDF IN MAP/REDUCE FRAME

The core idea of hadoop distributed computing is partitioning task, running concurrently. From the tf-idf formula, we can see, it is very suitable for distributed computing to solve.Term frequency is only related to the number of the term occurrences in the document and the total number of terms in this document. Thus, by partitioning the data, we can calculate in concurrent and distributed way to accelerate the speed of word frequency statistics. After calculating the term frequency, tf-idf weight calculation will depend on the number of documents containing the word (because the total number of documents is a constant). So, we can also calculate tf-idf in parallel if the number of documents containing the word is known.In this paper, we design three MapReduce process to achieve tf-idf calculation.

A. calculate the number of occurrences of the word in document

In the mapper, we use regular expressions to match words and write <<word#documentName>, 1> pairs to intermediate values which will be processed by reducer. Then we calculate the number of occurrences of the word in

document directly in the reducer. The output of reducer need to be written to the intermediate files (tempFile1) which will be processed in next MapReducer process. The output is using <word # documentName> as the key, <n> as the value. 'n' is the number of occurrences of the term 'word' in the 'documentName'. Function is designed as follows:

Map():
Input: <documentLineNumer, contents>
Output: <<word#documentName>, 1>
Reduce():
Input: <<word#documentName>, 1>
Output: <<word#documentName>, n>

B. calculate the total number of words of each document

In this step, we reorganized the <key,value> pairs in mapper(using documentName as key and <word=n> as value). Then we calculate the total number of words of each document in reducer. The output of reducer need to be written to the intermediate files (tempFile2) which will be processed in next MapReducer process. The output is using <word # documentName> as the key, <n/N> as the value. 'n' is the number of occurrences of the term 'word' in hte document 'documentName', and 'N' is the total number of words of 'documentName'. Function is designed as follows:

Map():
Input: < <word#documentName>, n >
Output: <documentName,< word=n> >
Reducer():
Input: <documentName,< word=n> >
Output: <<word#documentName>, <n/N>>

C. calculate TF-IDF

In this step, we reorganized the <key,value> pairs in mapper(using word as key and<documentName#n/N> as value). Then we calculate the number 'd' which is the number of documents containing this word and the number 'D' which is the total number of whole documents. At last, we can calculate the TF-IDF according to formula TF-IDF = n / N * log (D / d) .Function is designed as follows:

Map():
Input: <<word#documentName>, n/N>
Output: <word, <documentName#n/N>>
Reducer():
Input: <word, <documentName#n/N>>
Output: <<word#documentName>, n / N * log (D / d) >

D. The whole process flow

V. Experiments and analysis

A. Experimental data: We download 200,000 Chinese documents from the text classification corpus provided by Sogou laboratory as test corpus.

B. Data preprocessing: Split documents by using open source Chinese word segmentation tools IKAnalyzer and remove stop words at the same time. Management too many small files will reduce the efficiency hadoop.So we need to archive the dataset. The final test data are as follows:

C. The hadoop cluster setup: We build a cluster by using five computers. We use one machine as master which is responsible for scheduling Job and managing file namespace. And the rest of machines are responsible for calculating tf-idf and storage files.

D. The test result

We assess tf-dif applying Map/Reducer frame with traditional tf-idf. Traditional tf-idf is run in a machine, and it can't run in concurrent and distributed way.

From the figure, we can see,when dealing with small dataset(<200MB), the gap between the old algorithm and the new algorithm is not obvious. This is because that the maintenance and network transmission of hadoop itself requires some resource consumption. With the increment of data, the calculating time of traditional tf-idf sharply increases. But after applying hadoop, the calculating time increase linearly with the increment of data, and is far less than the former.

VI. Summaries

This article makes use of the service provided by hadoop, and improves the efficiency of traditional tf-idf algorithm. If we want to continue to improve the efficiency, we can use SequenceFile to deal with input [7], compress the output of Mapper and Reducer or reduce the generation of intermediate files. In addition, we can expand the scale of the Hadoop cluster and change the parameters of the cluster.

REFERENCES

[1] SALTON G, BUCKLEY C. Term-weighting approaches in automatic text retrieval [J]. Information Processing and Management, 1988, PP513 - 523.

[2] SALTON G, CLEMENT T Y. On the construction of effective vocabularies for information retrieval[C].Proceedings of the 1973 Meeting on Programming Languages and Information Retrieval. New York: ACM, 1973: 11.

[3] Hadoop, http://hadoop.apache.org/hdfs/.

[4] Ralf Lammel, Data Programmability Team. Google's MapReduce Programming Model-Revisited[R]. Redmond, WA, USA: Microsoft Corp. 2007.

[5] Owen O'Malley. Programming with Hadoop's Map/Reduce[R]. ApacheCon EU, 2008.

[6] Jeffrey Dean, Saniay Ghemawat. MapReduce: Simplified Data Processing on Large Ousters[C]. OSDI'04: Sixth Symposium on Operating System Design and Implementation, San Francisco, CA, December, 2004.

[7] Tom White. Hadoop: The Definitive Guide[M]. O'Reilly,2010.5.

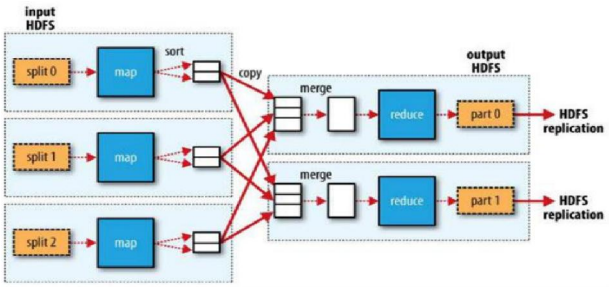

Figure 1. The relationship between HDFS and MapReduce

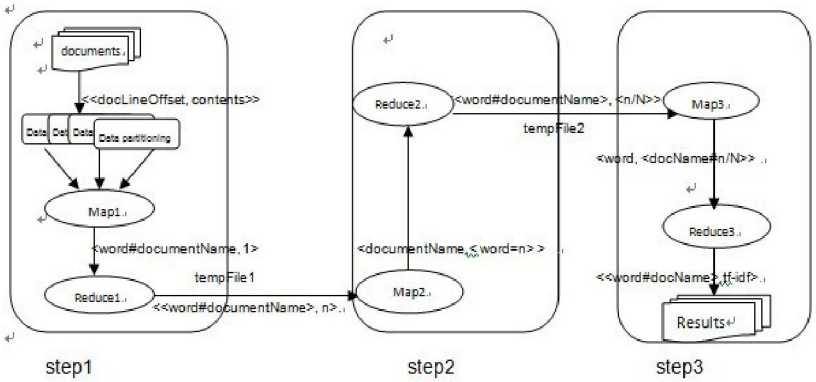

Figure 2. the whole process flow

TABLE I. TEST DATA

Dataset	Size(MB)	Number of Document
S1	40	10000
S2	80	20000
S3	160	40000
S4	320	80000
S5	520	130000
S6	780	200000

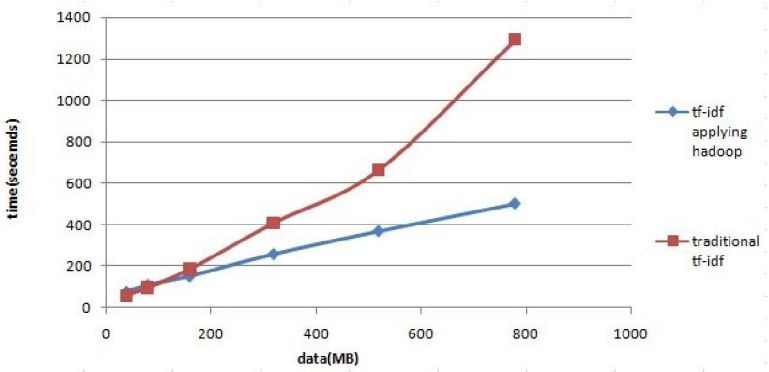

Figure 3. the Comparison between old and new tf-idf