ISBN 978-952-5726-07-7 (Print), 978-952-5726-08-4 (CD-ROM)
Proceedings of the Second Symposium International Computer Science and Computational Technology(ISCSCT '09)
Huangshan, P. R. China, 26-28,Dec. 2009, pp. 278-280

Design and Implement of Distributed Document Clustering Based on MapReduce

Jian Wan, Wenming Yu[1], and Xianghua Xu
Grid and Services Computing Lab
School of Computer Science and Technology
Hangzhou Dianzi University, Hangzhou 310037, China
[1]yuwenming_001@163.com

***Abstract*—In this paper, we describe how document clustering for large collection can be efficiently implemented with MapReduce. Hadoop implementation provides a convenient and flexible framework for distributed computing on a cluster of commodity machines. The design and implementation of tfidf and K-Means algorithm on MapReduce is presented. More importantly, we improved the efficiency and effectiveness of the algorithm. Finally, we give the results and some related discussion.**

***Index terms*—MapReduce, tfidf, K-Means clustering**

I. INTRODUCTION

With the rapid development of the Internet, huge volumes of documents need to be processed in a short time. Research on web mining focuses on scalable method applicable to mass documents[1]. Storage and computing of mass documents data in a distributed system is an alternative method[2]. In distributed computing, a problem is divided into many tasks, each of which is solved by one computer. However, many problems such as task scheduling, fault tolerance and inter-machine communication are very tricky for programmers with little experience with parallel and distributed system.

In this paper we describe our experiences and findings of document clustering based on MapReduce. MapReduce [3] is a framework which programmers only need to specify Map and Reduce functions to make a huge task parallelize and execute on a large cluster of commodity machines. In the document pre-processing stage, we design a new iterative algorithm to calculate tfidf weight on MapReduce in order to evaluate how important a term is to a document in a corpus. Then, a K-Means clustering is implemented on MapReduce to partition all documents into k clusters in which each documents belongs to the cluster with the same meaning. More importantly, we find that ignoring the terms with the highest document frequencies can not only speed up our algorithm on MapReduce, but also improve the precision of document clustering slightly. Experiments show that our method in approximately linear growth in required running time with increasing corpus size for corpus containing several ten thousand documents.

II. MAPREDUCE AND HADOOP

Many real world tasks have the same characteristics: a computation is applied over a large number of records to generate partial results, which are then aggregated in some fashion. MapReduce is a programming model which is specializing in handing problems having "Divide and Conquer" structure. MapReduce inspired by functional language consists of the Map and Reduce abstract concepts. A map function process each logical "record" in our input in order to compute a set of intermediate key/value pairs, and then a Reduce function accepts an intermediate key and a set of values for that key in order to combine the derived data appropriately. Figure 1 illustrates the two processing stages.

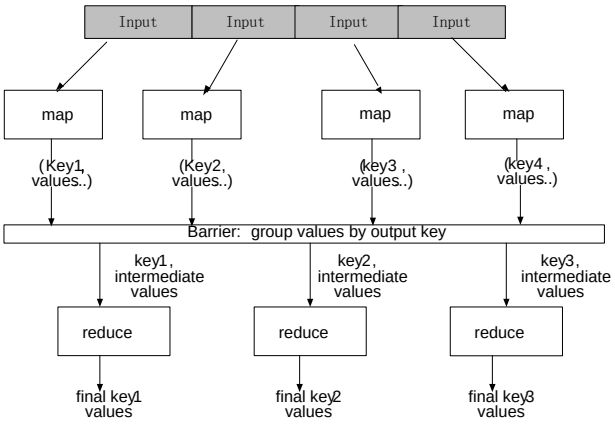

Figure1. Processing procedure of MapReduce

Apache Hadoop [4] is a Java software framework that consists of a MapReduce model and a distributed file system (HDFS, similar to GFS[5]). HDFS is designed to scale to mass storage across multiple machines, and transparently provides read/write, backup and fault-tolerance for users. Hadoop is becoming increasingly popular[6, 7] because it hides the messy details of parallelization, fault-tolerance, data distribution and load balancing.

III. CALCULATE TFIDF ON MAPREDUCE

The tfidf weight [8](term frequency–inverse document frequency) is a weight often used in text mining and information retrieval. The importance of a term increases proportionally to the number of times a term appears in the document but is offset by the frequency of the term in the corpus. Therefore the weight of feature term t_i in the corpus can be calculated using the classic tfidf scheme in formula (1).

$$w_{ij} = tf_{ij} \times idf_i = |t_i| / |d_j| \times \log(N / n_i) \qquad (1)$$

f_{ij} is the frequency of feature term t_i in the document d_j. It can be designated $|t_i|/|d_j|$, where $|t_i|$ is the number of occurrences of the term t_i in document d_j, and $|d_j|$ is the total number of terms in document d_j. N is the total number of documents in the corpus, and n_i is the number of documents that contains term t_i. We can see from the formula (1) that we should calculate $|t_i|$, $|d_j|$ and n_i on MapReduce to get tfidf.

1) Number of times a term t_i appears in a given document: The format of input data to Map function is (Docname, content), which means that document name is key and relevant content is value. For each term in the document, Map function output ((term, Docname), 1) which means this term occurrences one time in this document. Reduce functions accepts the output of former Map functions, and aggregate the records with the same key. The output format of Reduce functions is ((term, Docname), $|t_i|$). In practice, we can add a Combiner function to accelerate the computing speed. The function of Combiner function is the same as the Reduce function.

2) Number of terms in each document: This step's input data is the output of the first step, and the map functions convert the format into (Docname, (term, $|t_i|$)). The Reduce functions get the records sharing the same docname, and accumulate the number of different terms $|t_i|$ into $|d_j|$ in the same document. The output format of this step is ((term, Docname), ($|t_i|$, $|d_j|$))。

3) Number of documents term t_i appears in: The Map function in this step turn the output of above step into the format of (term, (Docname, $|t_i|$, $|d_j|$, 1)), which means that this term appears in one document. The Reduce function accumulate "1" with the same term into n_i, this is the number of documents contain the term t_i。 The output format of this step is ((term, Docname), ($|t_i|$, $|d_j|$, n_i)).

4) *Calculate tfidf :* The output of step 3 means that $|t_i|$ is the occurrence time of term t_i in document d_j, $|d_j|$ is the number of all terms in document d_j and n_i is the number of documents contain the term t_i. We can just use formula (1) to calculate tfidf of terms in different documents. The output format of the result is $(Docname, (term_1 \& tfidf_1, \cdots, term_n \& tfidf_n))$.

IV. K-MEANS CLUSTERING

K-Means clustering [9] choose k initial points and mark each as a center point for one of the k sets. Then for every item in the total data set it marks which of the k sets it is closest to. It then finds the average center of each set, by averaging the points which are closest to the set. With the new set of centers (centroid), it repeats the algorithm until convergence has been reached.

The implementation of document clustering on MapReduce accepts two input directories: one is the documents directory with the output of calculating tfidf, and one is centers directory with k initial document centers. The k initial document centers are chosen from the records of documents directory. Note that the k-line document data have the same terms as fewer as possible.

In every iteration, the MapReduce framework will partition the input files of document directory into a set of M splits, and then these splits are processed in parallel by M Map functions. Map functions read in a document with the format $(Docname, (term_1 \& tfidf_1, \cdots, term_n \& tfidf_n))$. Map functions should determine which of the current set of k document centers (in centers directory) the document is closest to and emits a record containing all the document's data and its chosen k-center with the format $(k-center, (Docname, term_1 \& tfidf_1, \cdots, term_n \& tfidf_n))$. The Reduce function receives a k-center and all documents which are bound to this k-center. It should calculate a new k-center, and put the new k-center in centers directory. To evaluate the distance between any two documents, we use the cosine similarity metric of tfidf, and use arithmetic average to calculate the new k-center. Note that the contents in document directory will not change during the process. The whole K-Means clustering on MapReduce can be expressed as the following two step:

$$map: (Docname, term \& tfidf\_list) \rightarrow (k\_center, (Docname, term \& tfidf\_list)) \quad (2)$$

$$reduce: (k\_center, (Docname, term \& tfidf\_list)) \rightarrow (new\_k\_center, term \& tfidf\_list) \quad (3)$$

V. EXPERIMENT EVALUATIONS

In our experiment, we used Hadoop version 0.18.2 running on a cluster with 5 machines (one is the master, also act as a slave). Each machine has two single-core processors (running at 2.33GHz), 1GB memory, and 130GB disk, and the network bandwidth is 100Mbps. The configuration of Hadoop is two map functions running on a processor core simultaneously. The number of map functions can be controlled precisely.

We used the Sogou documents classification corpus[1], containing 80k documents, totaling approximately 211MB. There are 10 different subjects in the corpus, and 8k documents in each subject. These documents are parsed and terms are stemmed. All empty words (we maintain a Chinese stop word list) in these documents are removed.

We always use running time to measure the efficiency. One issue that became evident in initial experiments was the prevalence of "stragglers", which means one or two reducers that take significantly longer

[1] http://www.sogou.com/labs/dl/c.html

than the others (this is a common problem, see) due to the Zipfian distribution of terms. In our experiment, we eliminate the terms with the highest document

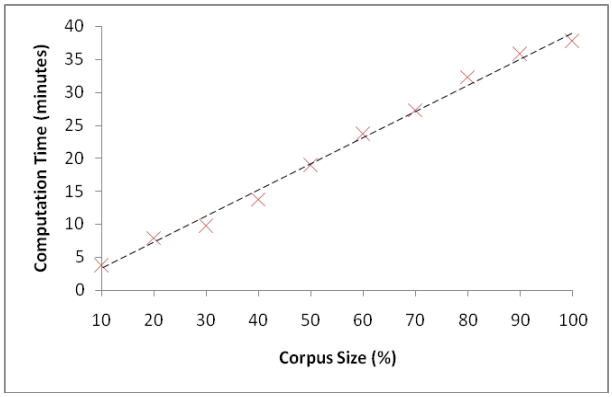

Figure2. Running time of tfidf algorithm with the different size of collections

frequencies. We adopt a 95% cut at step 3 in section 3, which means that the most frequent 5% of terms were ignored. This method greatly increases the efficiency of our algorithm on Hadoop. On the other hand, because the terms we discarded are non-discriminative, the precision of document clustering is improved slightly.

Figure 2 shows the running time of tfidf algorithm on our cluster with increasing collection size for collection containing 80k documents. We get the result just to see the effect of our algorithm on Hadoop, don't configure our cluster in optimizing. We find the time used for calculating the whole collection is more than half an hour. However, the running time and space required is approximately linear with the size of collection, this is characteristics we expect in processing mass data.

We measure the running time of K-Means clustering on the clusters, and then implement conventional K-Means on single machine as a benchmark. These results were compared against an equivalent run on the machine with the same times of iterations (5 times). We can see from Figure 3 that the running time of K-Means algorithm on a cluster with 5

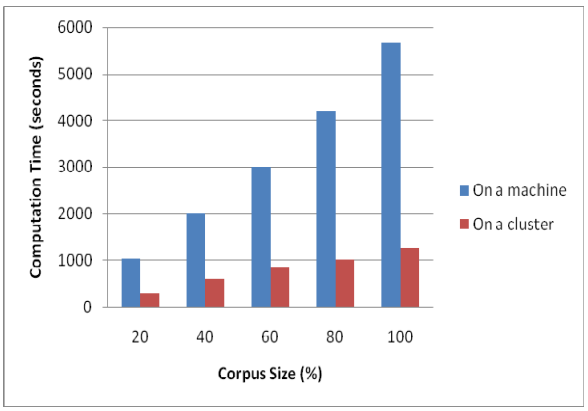

Figure 3. Running time comparison of K-Means on different size of collections

machines relatively much less than on a single machine. More importantly, with the size increasing of our collections, the advantage of efficiency has become increasingly evident.

VI. Conclusions

The paper has introduced a mass documents clustering in a distributed system Hadoop. Experiments show the scalability of our method in processing mass data. The contributions of our work lie in both design and implement tfidf and K-Means algorithm on MapReduce. We believe that our work provides an example of a programming paradigm that could be useful for a broad range of text analysis problems. Finally, we always pay attention to the alternative approaches to similar problems based on MapReduce [10]. Hadoop provides unprecedented opportunities for researchers to handle real-world problems at scale.

Acknowledgment

This paper is supported by National Science Foundation of China under grant No.60873023, and Science and Technology R&D Program of Zhejiang Province, China under grant No. 2008C13080, No.2007C21G3230005.

References

[1] Roberto, J.B., M. Yiming, and S. Ramakrishnan, Scaling up all pairs similarity search, in Proceedings of the 16th international conference on World Wide Web. 2007, ACM: Banff, Alberta, Canada.

[2] Michael, J.F., Evolution of distributed computing theory: from concurrency to networks and beyond, in Proceedings of the twenty-seventh ACM symposium on Principles of distributed computing. 2008, ACM: Toronto, Canada.

[3] Jeffrey, D. and G. Sanjay, MapReduce: simplified data processing on large clusters. Commun. ACM, 2008. 51(1): p. 107-113.

[4] Apache Lucene Hadoop[EB/OL].http://hadoop.apache.org/.

[5] Sanjay, G., G. Howard, and L. Shun-Tak, The Google file system, in Proceedings of the nineteenth ACM symposium on Operating systems principles. 2003, ACM: Bolton Landing, NY, USA.

[6] Jimmy, L., Brute force and indexed approaches to pairwise document similarity comparisons with MapReduce, in Proceedings of the 32nd international ACM SIGIR conference on Research and development in information retrieval. 2009, ACM: Boston, MA, USA.

[7] ZHENG Xin-jie, ZHU Cheng-rong, and X. Qi-bang, Design and Implementation of Distributed Ray Tracing Using MapReduce. Computer Engineering, 2007(22).

[8] Salton, G. and T.Y. Clement, On the construction of effective vocabularies for information retrieval, in Proceedings of the 1973 meeting on Programming languages and information retrieval. 1973, ACM: Gaithersburg, Maryland.

[9] Fasulo, D., An Analysis of Recent Work on Clustering Algorithms, in Technical Report UW-CSE-01-03-02. 1999, ACM: University of Washington.

[10] Chu, C.-T., S.K. Kim, and Y.-A. Lin, Mapreduce for machine learning on multicore, in In Proceedings of Neural Information Processing Systems Conference (NIPS) 2007.