Term Frequency-Inverse Document Frequency on Husky

Wu Jiayi
Department of Computer Science and Engineering
the Chinese University of Hong Kong
Email: 1155046964@link.cuhk.edu.hk

***Abstract*—Husky is a data-parallel computing system which was expected to better balance high performance and low development cost. In this summer research internship, I developed an efficient, scalable and distributed implementaion of term frequency-inverse document frequency (TF-IDF) on Husky.**

***Keywords*—Husky, TF-IDF, distributed, machine learning.**

I. Introduction

Husky [2] is developed mainly for in-memory large scale data mining, and also serves as a general research platform for designing efficient distributed algorithms. I implemented Term frequency-inverse document frequency (TF-IDF [1]) algorithm on it in this summer. TF-IDF is a numerical statistic widely used in text mining to reflect the importance of a term to a document in the corpus. An important factor to the importance of a term to a document is the number of times a term occurs in a document, which is called its term frequency. In formula, TF = (Number of times term t appears in a document) / (Total number of terms in the document). It is obvious that in the same corpus, if a term occurs more in a document that in another, the term is more important and has a bigger term frequency in the former document. However, term frequency itself is not a good statistics because common words, such as ”the”, ”a” and ”for”, will always have big term frequency values but usually are not helpful to find out the topics of the documents. So, TF-IDF also includes another statistics, the inverse document frequency, which measures whether the term is common or rare across all documents. It is obtained by dividing the total number of documents by the number of documents containing the term, and then taking the logarithm of that quotient. In formula, IDF = log(Total number of documents / Number of documents with term t in it). For example, if a term occurs in every document in the corpus, the inverse document frequency would be 0. For a specific term, the more documents contain it, the more common it is and the smaller inverse document frequency it has. TF-IDF combines these two factors and is defined as the product of term frequency and inverse document frequency.

II. Implementation

A. Step 1

In this implementation, there are two object lists, the document list and the term list. The documents are stored in the document list after being loaded. They are stored on different workers, but it is still easy to calculate term frequency of each word to each document directly because the calculation only requires information for one document and the workers do not need to exchange information. Each document has a vector with each non-duplicated term in the document and a vector with corresponding term frequency. Since each document need to be dealt with in this step, an aggregator can be used to count the total number of documents. During the calculation, each document will also send messages to the terms in the document which are members in the term list.

B. Step 2

Once a message is sent to a term that the term list does not contain, the term will be added to the term list. After a term has been added to the term list, it can know how many documents contain it by the number of messages it received. Then the term can get its inverse document frequency since both the number of documents containing it and the total number of documents have been counted. As introduced in the algorithm part, IDF = log(Total number of documents / Number of documents with term t in it), which is a static value to a term and independent of which documents the term is in.

C. Step 3

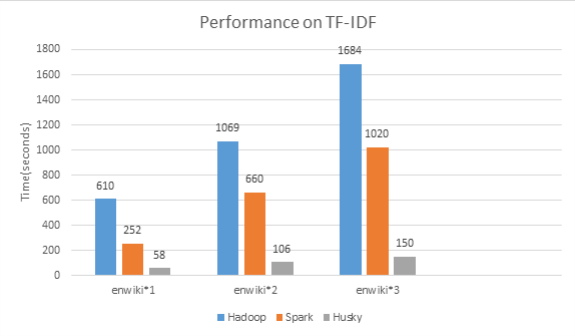
The term frequency of each term to each document and the inverse document frequency have already been calculated, so the remaining step is to combine these two statistics and get TF-IDF value of each term to each document. Here, a mechanism different from sending messages in the step 1 is used. In step 1, since workers do not know which worker the term that a message is sent to is on, the term list is globalized among all workers which cause much communication. However, since the term does not exist as an object before receiving a message, it must be globalized. Here, the objective is to enable document list get inverse document frequency values. Since each document has already in the document list, it is not necessary to globalize document list. Instead, each document send requests to all terms in it and the ids of the requests are same as the terms. Then each term broadcasts its inverse document frequency value as its reply. The term does not need to know which document requests its reply and each request sent by a document can find a corresponding reply by finding the reply given by the term with the same id with itself. Since each term for each document corresponds a request with has the inverse document frequency value of the term as its reply, it can get its TF-IDF value by multiplying the inverse document frequency value to the term frequency value calculated in step 1. Now the TF-IDF values of terms in a document can be stored as a vector of the document as the way term frequency values are stored in step 1.

D. Step 4

The implementation offers a function which enables users to check the TF-IDF value of a given word in a given document. This implementation also enables users to get the list name of the document list so that the user could get all the information stored in the document list.

III. Performance

The dataset is the English Wikipedia corpus, denoted by "en-wiki * 1", which contains over 4.8 million documents and 1.7 billion terms. I also tested heavier workloads by duplicating "enwiki * 1" by 2 and 3 times, denoted by "enwiki * 2" and "enwiki * 3".



As shown in the figure, Spark is 39% to 58% faster than Hadoop, but Husky is even around 5 times faster than Spark. The advantage is more signicant when the datasets get larger.

IV. Conclusion

Here, I introduced some basic APIs of Husky and TF-IDF algorithms. I have presented my distributed implementation of TF-IDF and showed that it is efficient and scalable.

Acknowledgment

I would like to express my special thanks of gratitude to Professor James Cheng who gave me the golden opportunity to do this wonderful project. I would also like to thank Fan Yang, Jinfeng, Yuzhen, Yunjian, Legend and all the teammates for the great and patient help in the summer research internship.

References

[1] H. C. Wu, R. W. P. Luk, K. Wong, and K. Kwok. Interpreting TF-IDF term weights as making relevance decisions. *TOIS*, 26(3), 2008.

[2] F. Yang, J. Li, and J. Cheng. Husky: Towards a more efficient and expressive distributed computing framework. *PVLDB*, 9(5):420–431, 2016.