International Journal of Grid and Distributed Computing
Vol. 9, No. 4 (2016), pp.95-100
http://dx.doi.org/10.14257/ijgdc.2016.9.4.09

Large-Scale Text Similarity Computing with Spark

Xiaoan Bao[1], Shichao Dai[1], Na Zhang[1*], Chenghai Yu[1]

The institute of software of Zhejiang Sci-Tech university
zhangna@zstu.edu.cn

Abstract

Text understanding is a hot research in Natural Language Processing and Information Retrieval. In recent years, it has received wide attention and research. In the era of big data, Understanding text in large-scale datasets is a challenge. Although the earliest systems designed for these workloads, such as MapReduce, gave users a powerful, but low-level, procedural programming interface. So, MapReduce doesn't compose well for lager text applications. Recently, Spark, an in-memory cluster-computing platform, has been proposed. It has emerged as a popular framework for large-scale data processing and analytics. It provides a general-purpose efficient cluster computing engine and simpler for the end users. In this work, we consider using Vector Space Model (VSM) and TF-IDF weighting schema and feature hashing feature extraction techniques in order to solve the problem of large-scale text data similarity computing by Spark. As a result, Experimental results that using Spark in order to solve document similarity computation problems as soon as quickly by 20Newsgroups. In additions, It is more benefit from document classification and clustering of machine learning tasks.

Keywords*: Text Similarity; Vector Space Model; TF-IDF; MapReduce; Spark*

1. Introduction

With the rapid growth of Internet Information and entering the era of big data, how to deal with massive text data has become the current Information Retrieval and Natural Language Processing field widely research problems. For large-scale text similarity computing is an important research hotspot. In the real world, the text is the most important carrier of information. However, the text similarity computing is a very basic and key problem in information processing. The text similarity computing has been widely used for Information Retrieval, Information Filtering, Knowledge Mining and Machine Translation fields.

MapReduce [1] is a programming model for large-scale data processing developed by Google company, which is the general computational framework for large-scale text data processing, with simple programming and easy to expand, good fault tolerance. In addition to Google, MapReduce has many versions, one of the most classic is Hadoop [2] , a java language based on open source distributed computing framework that provides the core to the MapReduce programming interface and Distributed File System HDFS(Hadoop distributed File System) [3] , able to handle millions of real ZB data. But more and more studies have proved that, Hadoop's MapReduce programming model is relatively simple, but the core calculation for the amount of data processing job is not complicated, but for the more complex model, such as iteration, recursion, iteration machine learning algorithms using MapReduce programming to achieve not only complex, but also very difficult to satisfactory processing efficiency. Nevertheless, the heavy disk I/O of reloading the data at each MapReduce operation is an obstacle of developing efficient iterative machine learning algorithms on this platform. [1]

Na Zhang is the corresponding author.

ISSN: 2005-4262 IJGDC

Recently, Spark [4] has been considered as a great alternative of MapReduce in overcoming the disk I/O problem. Spark is an in-memory cluster-computing platform that allows the machines to cache data in memory instead of reloading the data repeatedly from disk.

In this paper, we consider using Vector Space Model (VSM) [5] and TF-IDF [6] weighting schema and feature hashing feature extraction techniques [11] in order to solve the problem of large-scale text data similarity computing by Spark.

This paper is organized as follows. At First briefly introduces Spark (§2), We then describe Vector Space Model (VSM) and text similarity computing method (§3), and Experimental results in §4. Finally, §5 concludes this work.

2. Apache Spark

Traditional MapReduce frameworks such as Hadoop have inefficient performance when conducting iterative computations because it requires disk I/O of reloading the data at each iteration. To avoid extensive disk I/O, distributed in-memory computing platforms become popular. Apache Spark is a general-purpose in-memory cluster computing engine with APIs in Scala, Java and Python and libraries for data science, streaming, graph processing and machine learning. [4] Released in 2010, it is to our knowledge one of the most widely-used system with a "language integrated" API similar to DryadLINQ [7] , and the most active open source project for big data processing. Spark had over 400 contributors in 2014, and is packaged by multiple vendors.

Spark allows user to develop applications by high-level APIs. It enables the machines to cache data and intermediate results in memory instead of reloading them from disk at each iteration. In order to support parallel programming, Spark provides resilient distributed datasets (RDDs) and parallel operations. [16] This section discusses the details of these techniques.

2.1. Resilient Distributed Datasets

The core of Spark is a concept called the Resilient Distributed Datasets (RDDs) [10][12][15]. An RDD is a collection of "records" (strictly speaking, object of some type) that is distributed or partitioned across many nodes in a cluster. RDDs can be created from existing collections and from Hadoop-based input sources, including the local filesystem, HDFS, and Amazon S3 [3] . Once we have created an RDD, we have a distributed collection of records that we can manipulate. In Spark's programming model, operations split into transformations and actions. Transformations, including operations like map and filter, create a new RDD from existing one. Actions, such as reduce and collection, conduct computions on RDD and return the result to the driver program. Fox example, the Scala [14] code below counts starting with "Sparks" in a text file:

```
val conf = new SparkConf ().setAppName("word count")

val sc = new SparkContext(conf)

val lines = sc.textFile("hdfs://....")

val errors = lines.filter(s => s.contians("Sparks"))

println(errors.count())
```

This code creates an RDD of strings called lines by reading an HDFS file, then transforms it using filter to obtian another RDD, Sparks. It then performs a count on this data.

One note about the API is that RDDs are evaluated lazily. Each RDD represents a "logical plan" to compute a dataset, but Spark waits until certain output operations, such as count, to launch a computation. For instance, in the example above, Spark will reading

lines from the HDFS file with applying the filter and computing a running count, so that it never needs to materialize the intermediate lines and Sparks results. [16]

2.2. Broadcast Variables and Accumulators

Another core feature of Spark is the ability to create two special types of variables: broadcast variables and accumulators. A broadcast variables is a read-only variable that is made available from the driver program that runs the SparkContext object to nodes that will execute the computation. This is very useful in applications that need to make the same data available to the worker in an efficient manner, such as machine learning algorithms. An accumulator is also a variable that is broadcasted to work nodes. The Key difference between a broadcast variable and an accumulator can be added to. There are limitations to this, that is, in particular, the addition must be an associative operation so that the global accumulated value can be correctly computed in parallel and returned to the driver program. Each worker node can only access and add to its own local accumulator value, and only the driver program can access the global value. Accumulators are also accessed within the Spark code using the value method.

2.3. Lineage and Fault Tolerance of Spark

RDDs are fault-tolerant by logging the transformations used to build a dataset rather than the actual data. The transformations operations are maintained as a lineage, which records how RDDs are derived from other RDDs. If a partition of an RDD is lost, the RDD has enough information about how it was derived from other RDDs to recompute just that partition. Thus, lost data can be recovered, often quite quickly, without requiring costly replication. They make the recomputation of RDDs efficient. The RDDs ensure that after reconducting the operations records in the lineage, we can obtain the same result.

3. Vector Space Model and Similarity Computing Techniques

Vector Space Model (VSM) was developed by the SMART information retrieval system (Salton, 1971) by Gerard Salton and his colleagues (Salton, Wong, & Yang, 1975). SMART pioneered many of the concepts that are used in modern search engines (Manning, Raghavan, 2008). [8][9]The VSM is commonly used in natural language processing model.

3.1. Text Representation

In the vector space model, a document is formalized as a vector in n dimensional space, each dimension of the vector is composed by the weights of feature items and item, feature weights using TF-IDF[5][6] method to calculate(1):

$$w(t_i,d)=\frac{tf(t_i,d)*\log\left(N/n_i+0.01\right)}{\sqrt{\sum_{t_i\in d}\left[tf(t_i,d)*\log\left(N/n_i+0.01\right)\right]^2}} \quad (1)$$

Among that: $w(t_i,d)$: the weight of feature items t_i in document d; $tf(t_i,d)$: feature items t_i appear in the document d frequency representation; N: the total number of training texts; n_i: Training text sets appear the text number of feature item t_i; The denominator for the normalization factor.

TF-IDF weighting method considering the different word frequency in the text and the words on the text of the resolution. In all text in the small number of words will give a high weight, can be regarded as a text classifier. TF-IDF is a calculation of weight based on statistics, In the global corpus text set comprises the sufficient number of cases, can get very good accuracy.

3.2. Similarity Computing Techniques

After getting the text feature vector, between text d_i and d_j similarity is calculated by computing the cosine distance to complete, as below (2):

$$Sim(d_i, d_j) = \cos\theta = \frac{\sum_{k=1}^{n} w_{ik} * w_{jk}}{\sqrt{\left(\sum_{k=1}^{n} w_{ik}^2\right)\left(\sum_{k=1}^{n} w_{jk}^2\right)}} \quad (2)$$

As shown in formula (2): to measure the similarity of text with two vector angle cosine. The bigger the Sim, that is to say, The higher the similarity degree between two texts.

The biggest advantage of the vector space model is that its huge advantage in knowledge representation, which provides the text of a mathematical approach. The text content into multi-dimensional vector space, the process of simplification of the text as a vector space vector operations, which greatly improves the computability and operability of natural language text.

4. Experimental Results

In this section, The experimental environments use Cloudera open-source cloud computing platform, Cloudera Manager 5.3.2 deploys hadoop and spark cluster, a control node (master node) and three computing nodes (Slave Node), the node configuration are the same, two core CPU (clocked: 2.3GHz), 4G of memory and 30G hard drive. Between control and compute nodes at Gigabit Ethernet connectivity, Hadoop version 2.5.0, Spark version 1.2.0, jdk version 1.7, scala version 2.10.4. In this experimental environment, we use a well-known text dataset called 20 Newsgroups, This is a collection of newsgroup messages posted across 20 different topics. This dataset contains 61,118 words and 18,774 documents. At the same time, This data splits up the available data into training and test sets that comprise 60 percent and 40 percent of the original data, respectively. The Newsgroups is shown in Table 1.

Table 1. 20 Newsgroups

comp.graphics comp.os.ms-windows.misc comp.sys.ibm.pc.hardware comp.sys.mac.hardware comp.windows.x	rec.autos rec.motorcycles rec.sport.baseball rec.sport.hockey	sci.crypt sci.electronics sci.med sci.space
misc.forsale	talk.politics.misc talk.politics.guns Talk.politics.mideast	talk.religion.misc alt.atheism soc.religion.christian

Then, We evaluates the similarity between two documents based on the words in the documents. Using TF-IDF, We have transformed each document into a vector representation. As a result, We randomly choose two messages from the guns newsgroup

to be relatively similar to each other. After tokenization, removing stop words, training a TF-IDF model, analyzing the TF-IDF weightings [13] , We can see that the cosine similarity between the documents is around 0.03: 0.033873765436943766

By contrast, we can compare this similarity score to one computed between one of our guns documents and another document chosen randomly from the sci.crypt newsgroup, using the same methodology , then, The cosine similarity is significantly lower at 0.0063:

0.0062839379303594265

Finally, it is likely that a document from another politics-related topic might be more similar to our guns document than one from a sci-related topic. However, we would probably expect a mideast document to not be as similar as our guns document. Then, We computing the similarity between a random message from the mideast newsgroup and our guns document.

Indeed, we found that the mideast and guns documents have a cosine similarity of 0.02, which is significantly higher than the sci.crypt document, but also somewhat lower than the other guns document:

0.02400287495724198

5. Conclusion

In this work, we using Vector Space Model in order to transform each document into a vector representation and evaluate the similarity between two documents based on the words in the documents using spark. Therefore, we expect two document to be more similar to each other if they share many terms. Conversely, we expect two documents to be less similar if they each contain many terms that are different from each other. So, we compute cosine similarity by computing a dot product of the two vectors and each vector is made up of the terms in each document, we can see that documents with a high overlap of terms will tend to have a higher cosine similarity.

In short, by Spark Cluster Computing Framework, we are able to solve the problem of large-scale text date similarity computing. In additions, It is more benefit from document classification and clustering of machine learning tasks. However, we have not consider the similarity of two words in the sense of meaning. After work, we will continue to work about the similarity between two words, based on their meaning.

Acknowledgements

This research was supported in part by the National Natural Science Foundation of China (No.61379036); Major scientific and technological special key industrial projects of Zhejiang Province, China (Grant No. 2014C01047); the Natural Science Foundation of Zhejiang Province, China (No.Y13F020175, LY12F0204); 521 ta-lent project of Zhejing Sci-Tech University; the New-shoot Talents Program of Zhejiang province (Grant No.2014R406073); Public technology research plan of Zhejiang Province(Grant No. 2014C33102).

References

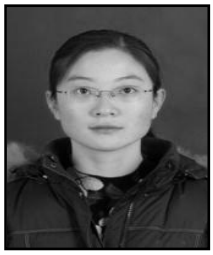
[1] J. Dean and S. Ghemawat, " MapReduce: Simplified data processing on large clusters". In OSDI, **(2004)**.
[2] T. While, "Hadoop: The Definitive Guide", 4th Edition. O'Reilly Media, **(2015)**.
[3] A. Holmes; "Hadoop in Practive", 2nd Edition. Manning Publications, **(2014)**.
[4] Apache Spark project. http://spark.apache.org.
[5] G. Salton, *et al*.. Vector-space model for automatic indexing [J]. Communications of the Acm, vol. 18, **(1975)**, pp. 613-620,
[6] S Gerard, and C Buckley. "Term-weighting approaches in automatic text retrieval." Information processing & management 24, no. 5 **(1988)**, pp. 513-523.
[7] M. Isard, M. Budiu, Y. Yu, A. Birrell, and D. Fetterly. Dryad: distributed data-parallel programs from sequential building blocks. In EuroSys '07, **(2007)**.
[8] P D Turney, and P Pantel. "From frequency to meaning: Vector space models of semantics." Journal of artificial intelligence research 37, no. 1, **(2010)**, pp.141-188.

[9] R. Vijay V., and SK Michael Wong. "A critical analysis of vector space model for information retrieval." Journal of the American Society for information Science 37, no. 5 **(1986)**, pp. 279-287.
[10] M. Zaharia, M. Chowdhury, M. J. Franklin, S. Shenker, and I. Stoica,"Spark: cluster computing with working sets,"in Proceedings of the 2nd USENIX conference on Hot topics in cloud computing, **(2010)**.
[11] W Kilian, *et al*. "Feature hashing for large scale multitask learning." Proceedings of the 26th Annual International Conference on Machine Learning. ACM, **(2009)**.
[12] M. Zaharia, M. Chowdhury, T. Das, A. Dave, J. Ma, M. McCauley, M. J. Franklin, S. Shenker, and I. Stoica, "Resilient distributed datasets: A fault-tolerant abstraction for in-memory cluster computing," in Pro-ceedings of the 9th USENIX conference on Networked Systems Design and Implementation, **(2012)**.
[13] R Collobert, J Weston, L Bottou, M Karlen, K Kavukcuoglu, P Kuksa,"Natural language processing (almost) from scratch" The Journal of Machine Learning Research 12, **(2011)**, pp. 2493-2537.
[14] Scala. http://www.scala-lang.org.
[15] M. Zaharia, "An Architecture for Fast and General Data Processing on Large Clusters", (PhD Disseration)

Authors

Bao Xiao'an, born in 1973, M.S. He is a professor and mainly researches adaptive software, software testing and intelligent information processing.

Dai Shichao, born in 1988, Master candidate.Her main research interests big data and system.



Zhang Na, born in 1977, M.S. She mainly researches software engineering and software testing technology.

Yu Chenghai , born in 1975,M.S. He mainly researches Software engineering and software testing technology.

