PPLSA: Parallel Probabilistic Latent Semantic Analysis Based on MapReduce

Ning Li[1,2,3], Fuzhen Zhuang[1], Qing He[1], and Zhongzhi Shi[1]

[1] The Key Laboratory of Intelligent Information Processing,
Institute of Computing Technology, Chinese Academy of Sciences, Beijing, China
[2] Graduate University of Chinese Academy of Sciences, Beijing, China
[3] Key Lab. of Machine Learning and Computational Intelligence, College of Mathematics and Computer Science, Hebei University, Baoding, China
{lin,heq}@ics.ict.ac.cn

Abstract. PLSA(Probabilistic Latent Semantic Analysis) is a popular topic modeling technique for exploring document collections. Due to the increasing prevalence of large datasets, there is a need to improve the scalability of computation in PLSA. In this paper, we propose a parallel PLSA algorithm called PPLSA to accommodate large corpus collections in the MapReduce framework. Our solution efficiently distributes computation and is relatively simple to implement.

Keywords: Probabilistic Latent Semantic Analysis, MapReduce, EM, Parallel.

1 Introduction

In many text collections, we encounter the scenario that a document contains multiple topics. Extracting such topics subtopics/themes from the text collection is important for many text mining tasks[1]. The traditional modeling method is "bag of words" model and the VSM(Vector Space Model) is always used as the representation. However, this kind of representation ignores the relationship between the words. For example, "actor" and "player" are different word in the "bag of words" model but have the similar meaning. Maybe they should be put into one word which means the topic. To deal with this problem, a variety of probabilistic topic models have been used to analyze the content of documents and the meaning of words[2]. PLSA is a typical one, which is also known as Probabilistic Latent Semantic Indexing (PLSI) when used in information retrieval. The main idea is to describe documents in terms of their topic compositions. Complex computation need to be done in the PLSA solving process. There is a need to improve the scalability of computation in PLSA due to the increasing prevalence of large datasets. Parallel PLSA is a good way to do this.

MapReduce is a patented software framework introduced by Google in 2004. It is a programming model and an associated implementation for processing and generating large data sets in a massively parallel manner [5,9]. Users specify a map function that processes a key/value pair to generate a set of intermediate key/value pairs, and a

Z. Shi, D. Leake, and S. Vadera (Eds.): IIP 2012, IFIP AICT 385, pp. 40–49, 2012.

reduce function that merges all intermediate values associated with the same intermediate key [5]. MapReduce is used for the generation of data for Google's production web search service, sorting, data mining, machine learning, and many other systems [5].

Two kinds of parallel PLSA with MapReduce have been propose in [10], which are P^2LSA and P^2LSA+ respectively. In P^2LSA, the Map function is adopted to perform the E-step and Reduce function is adopted to perform the M-step. Transferring a large amount of data between the E-step and the M-step increases the burden on the network and the overall running time. Differently, the Map function in P^2LSA+ performs the E-step and M-step simultaneously. However, the parallel degree is still not well. Different from these two algorithms, we have different parallel strategies. We design two kinds of jobs, one is for counting all the occurrences of the words and the other is for updating probabilities.

In this paper, we first present PLSA in Section 2. In Section 3 we introduce the MapReduce framework. We then present parallel PLSA (PPLSA). Section 5 uses large-scale application to demonstrate the scalability of PPLSA. Finally, we draw a conclusion and discuss future research plans in Section 6.

2 Probabilistic Latent Semantic Analysis

2.1 The Main Idea of PLSA

For extracting topics from the text collection, a well accepted practice is to explain the generation of each document with a probabilistic topic model. In such a model, every topic is represented by a multinomial distribution on the vocabulary. Correspondingly, such a probabilistic topic model is usually chosen to be a mixture model of k components, each of which is a topic[1]. One of the standard probabilistic topic models is the Probabilistic Latent Semantic Analysis (PLSA).

The basic idea of PLSA is to treat the words in each document as observations from a mixture model where the component models are the topic word distributions. The selection of different components is controlled by a set of mixing weights. Words in the same document share the same mixing weights.

For a text collection $D=\{d_1,\ldots,d_N\}$, each occurrence of a word w belongs to $W=\{w_1,\ldots,w_M\}$. Suppose there are totally K topics, the topic of document d is the sum of the K topics, i.e. $p(z_1 \mid d), p(z_2 \mid d),\ldots, p(z_K \mid d)$ and $\sum_{k=1}^{K} p(z_K \mid d)=1$. In other words, each document may belong to different topics. Every topic z is represented by a multinomial distribution on the vocabulary. For example, if the words such as "basketball" and "football" occur with a high probability, it should be considered that it is a topic about " physical education". Each w in document d can be generated as follows. First, pick a latent topic z_k with probability $p(z \mid d)$. Second, generate a word w with probability $p(w \mid z_k)$. Fig.1(a) is the graphic model and Fig.1(b) is the symmetric version with the help of Bayes' rule.

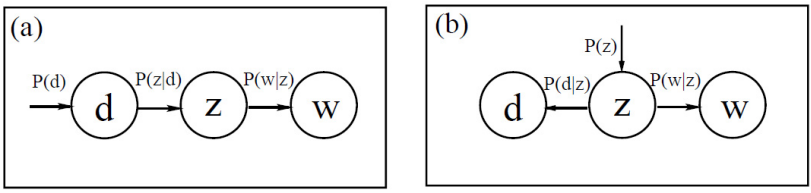


Fig. 1. The graphic model of PLSA

2.2 Solving PLSA with EM Algorithm

The standard procedure for maximum likelihood estimation in PLSA in the Expectation Maximization(EM) algotithm[3]. Acoording to EM algorithm and the PLSA model, the E-step is the following equation.

$$P(z \mid d, w) = \frac{P(z)P(d \mid z)P(w \mid z)}{\sum_{z'} P(z')P(d \mid z')P(w \mid z')} \tag{1}$$

It is the probability that a word w in a particular document d is explained by the factor corresponding to z.

The M-step re-estimation equations are as follows.

$$P(w \mid z) = \frac{\sum_{d} n(d, w)P(z \mid d, w)}{\sum_{d, w'} n(d, w')P(z \mid d, w')} \tag{2}$$

$$P(d \mid z) = \frac{\sum_{w} n(d, w)P(z \mid d, w)}{\sum_{d', w} n(d', w)P(z \mid d', w)} \tag{3}$$

$$P(z) = \frac{1}{R}\sum_{d, w} n(d, w)P(z \mid d, w), \quad R \equiv \sum_{d, w} n(d, w) \tag{4}$$

3 MapReduce Overview

MapReduce is a programming model and an associated implementation for processing and generating large data sets. As the framework showed in Fig.2, MapReduce specifies the computation in terms of a map and a reduce function, and the underlying runtime system automatically parallelizes the computation across large-scale clusters of machines, handles machine failures, and schedules inter-machine communication to make efficient use of the network and disks.

Essentially, the MapReduce model allows users to write Map/Reduce components with functional-style code.

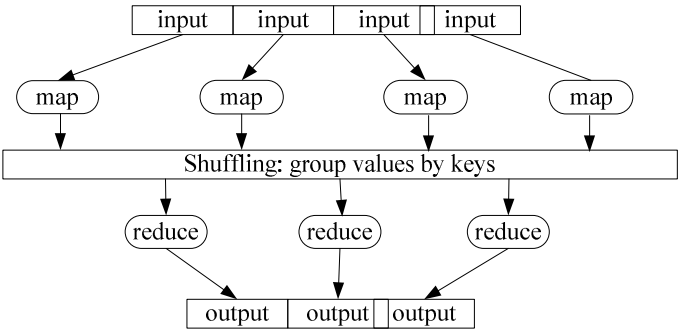


Fig. 2. Illustration of the MapReduce framework: the "map" is applied to all input records, which generates intermediate results that are aggregated by the "reduce"

Map takes an input pair and produces a set of intermediate key/value pairs. The MapReduce library groups together all intermediate values associated with the same intermediate key and passes them to the reduce function [6]. That is, a map function is used to take a single key/value pair and outputs a list of new key/value pairs. It could be formalized as:

$$map :: (key_1, value_1) \Rightarrow list(key_2, value_2)$$

The reduce function, also written by the user, accepts an intermediate key and a set of values for that key. It merges together these values to form a possibly smaller set of values. Typically just zero or one output value is produced per reduce invocation. The intermediate values are supplied to the users reduce function via an iterator. This allows us to handle lists of values that are too large to fit in memory. The reduce function is given all associated values for the key and outputs a new list of values. Mathematically, this could be represented as:

$$reduce :: (key_2, list(value_2)) \Rightarrow (key_3, value_3)$$

The MapReduce model provides sufficient high-level parallelization. Since the map function only takes a single record, all map operations are independent of each other and fully parallelizable. Reduce function can be executed in parallel on each set of intermediate pairs with the same key.

4 Parallel PLSA Based on MapReduce

As described in section2, EM algorithm is used to estimate parameters of the PLSA model. Our purpose is to compute $P(w \mid z)$, $P(d \mid z)$ and $P(z)$. The whole procedure is an iteration process. We set A to represent $n(d,w)P(z \mid d,w)$, and so the equation (2) to equation (4) can be rewritten as follows.

$$P(w \mid z) = \frac{\sum_d A}{\sum_{d,w'} A} \tag{5}$$

$$P(d \mid z) = \frac{\sum_w A}{\sum_{d',w} A} \tag{6}$$

$$P(z) = \frac{1}{R} \sum_{d,w} A, \quad R \equiv \sum_{d,w} n(d,w) \tag{7}$$

Note that the key step is to compute A and computing R is necessary for the computation of $P(z)$. In each iteration, computing $P(w \mid z)$ need to sum up A for each document d first and then do the normalization. Similarly, computing $P(d \mid z)$ need to sum up A for each word w. It is obviously that the computation of A for one document is irrelevant with the result of another one in the same iteration. For the same reason, computation of $\sum_w A$ and R is in the same situation. So the computation of A, $\sum_w A$ and R could be parallel executed. Therefore, we design two kinds of MapReduce job, one is to compute $\sum_d A$ and $\sum_w A$, the other is for computing R.

For the job computing $\sum_d A$ and $\sum_w A$, the map function, shown as **Map1**, performs the procedure of computing A and $\sum_w A$ for each document and thus the map stage realizes the computation of A and $\sum_w A$ for all the documents in a parallel way. The reduce function, shown as **Reduce1**, performs the procedure of summing up A to get $\sum_d A$.

For the job computing R, the map stage realizes the computation of $\sum_w n(d,w)$ for each document, and $R \equiv \sum_{d,w} n(d,w)$ is gotten in the reduce stage. The map function and reduce function are shown as **Map2** and **Recuce2** respectively.

As the analysis above, the procedure of PPLSA is shown in the following.

Procedure PPLSA

```
1. Input: Global variable numCircle, the number of latent
   topics
2. Initialize p(z), p(w|z), p(d|z);
3. The job computing R is carried out;
4. for (circleID = 1; circleID <= numCircle; circleID++)
5.    /*numCircle is the max number of interations.*/
6.      The job computing ∑_d A and ∑_w A is carried out.
7.      Compute ∑_{d,w} A;
8.      Update p(z), p(w|z), p(d|z);
9. end
10. Output p(z), p(w|z), p(d|z).
```

Map1 Map(*key*, *value*)

Input: pz, pdz, pwz, the offset *key*, the sample *value*
Output: <*key', value'*> pair

```
1. Initialize arrayzd[] which is used for storing ∑w A
2. for (int i = 1;i<strarry1.length;i++) /*strarry1.length=
   the number of words in value*/
3.     compute p(z)*p(d|z)*p(w|z) for each topic and proceed
   the normalization
4.     for(int j = 0;j<number of topics;j++)
5.       tmp= n(d,w)P(z|d,w)
6.       Output(key'=j+"-w-"+(WordID-1),value'=tmp);/*for
   computing P(w|z) */
7.       arrayzd[j] += tmp;    /*compute ∑w n(d,w)P(z|d,w) */
8.     end
9. end
10.  for (i=0;i<number of topics;i++)
11.       Ouput(key'=j+"-d-"+(DocID-1),value'=arrayzd[j]);
   /*for computing P(d|z) */
12.  End
```

Reduce 1 Reduce (*key*, *value*)

```
1. sum=0;
2. for(Text value:values)
3.    sum+=value /* sum up the values with the same key */
4. end
5. output(key,sum);
```

Map2 Map(*key*, *value*)

Input: the offset *key*, the sample *value*
Output: <*key', value'*> pair

```
1. nCount=0;
2. for (int i = 1;i< the number of words in value ;i++)
3.    get each word frequence freq[i];
4.    nCount+=freq[i];
5. end
6. Output(key'=a random number belong (0,100),
   value'=nCount)
```

Reduce 2 Reduce (*key*, *value*)

```
1. sum=0;
2. for(Text value:values)
3.    sum+=value;
4. end
5. output(key,sum);
```

5 Experimental Analysis

In this section, we evaluate the performance of PPLSA. Performance experiments were run on a cluster of 4 computers, each of which has four 2.8GHz cores and 4GB memory. Hadoop version 0.20.2 and Java 1.5.0_14 are used as the MapReduce system for all the experiments. Experiments were carried on 10 times to obtain stable values for each data point.

5.1 The Datasets

We performed experiments on two datasets: a subset of the TREC AP corpus containing 2246 documents with 10,473 unique terms and a dataset extracted from internet about stock, containing 316 html documents with 27,925 terms.

5.2 The Evaluation Measure

We use scaleup, sizeup and speedup to evaluate the performance of PPLSA algorithm.

Scaleup: Scaleup is defined as the ability of an m-times larger system to perform an m-times larger job in the same run-time as the original system. The definition is as follows.

$$Scaleup(data, m) = \frac{T_1}{T_{mm}} \tag{8}$$

Where, T_1 is the execution time for processing data on 1 core, T_{mm} is the execution time for processing m*data on m cores.

Sizeup: Sizeup measures how much longer it takes on a given system, when the dataset size is m-times larger than the original dataset. It is defined by the following formula:

$$Sizeup(data, m) = \frac{T_m}{T_1} \tag{9}$$

Where, T_m is the execution time for processing m*data, T_1 is the execution time for processing data.

Speedup: Speedup refers to how much a parallel algorithm is faster than a corresponding sequential algorithm. It is defined by the following formula:

$$Speedup = \frac{T_1}{T_p} \tag{10}$$

Where, p is the number of processors, T_1 is the execution time of the algorithm with one processor, T_p is the execution time of the parallel algorithm with p processors.

5.3 The Performance and Analysis

To demonstrate how well the PPLSA algorithm handles larger datasets when more cores of computers are available, we have performed scaleup experiments where the increase of the datasets size is in direct proportion to the number of computer cores in the system. We ran the datasets which are 60-times, 120-times and 240-times of the original ones on 4, 8, 16 distributed machines respectively. The scaleup performance of PPLSA is shown in Fig.3.

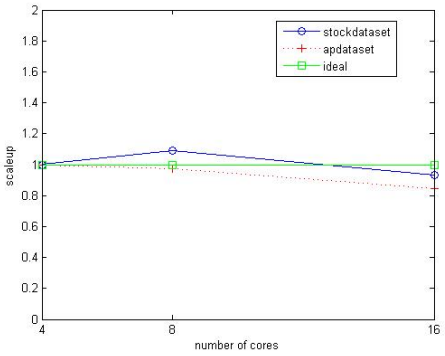
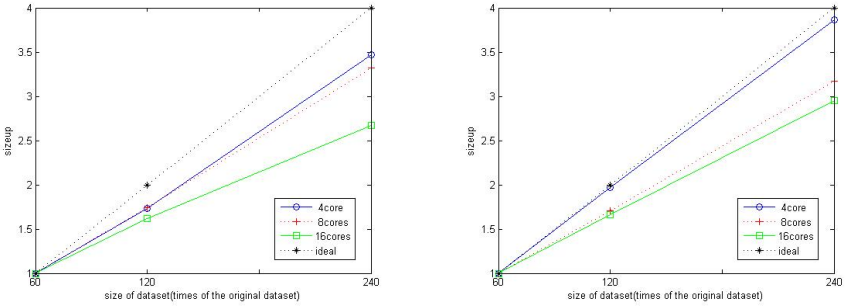


Fig. 3. Scaleup performance evaluation

We have plotted scaleup which is the execution time normalized with respect to the execution time for 4 machines. Clearly the PPLSA algorithm scales very well, being able to keep the execution time almost constant as the dataset and machine sizes increase.

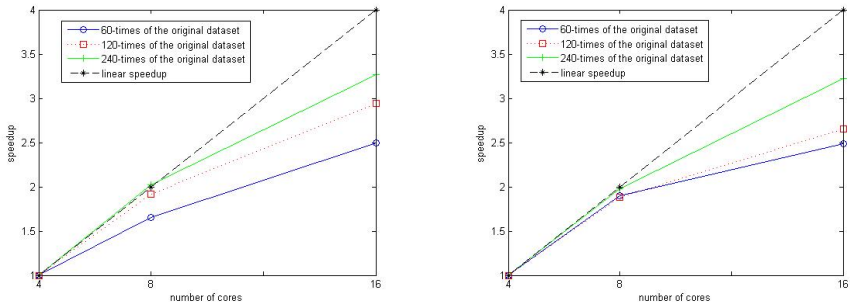
To measure the performance of sizeup, we fix the number of cores to 4, 8 and 16 respectively. Fig.4 shows the sizeup results on different cores. The results show sublinear performance for the PPLSA algorithm, the program is actually more efficient as the dataset size is increased. Increasing the size of the dataset simply makes the noncommunication portion of the code take more time due to more I/O and more documents processing. This has the result of reducing the percentage of the overall time spent in communication. Since I/O and CPU processing scale well with sizeup, we get sublinear performance.



(a) Sizeup for AP dataset (b) Sizeup for stock dataset

Fig. 4. Sizeup performance evaluation

We used 4 cores as the baseline to measure the speedup of using more than 4 cores. The speedup performances are shown in Fig.5.



(b) Speedup for AP dataset (b) Speedup for stock dataset

Fig. 5. Speedup performance evaluation

From Fig.5 we can see that PPLSA can achieve linear speedup when the core is small. However, the improvement becomes gradually undramatic as the number of processors grows. This is expected due to both the increase in the absolute time spending in communication between machines, and the increase in the fraction of the communication time in the entire execution time. When the fraction of the computation part dwindles, adding more machines (CPUs) cannot improve much speedup.

Moreover, the speedup performance shows better on the large datasets. This is an artifact of the large amount of data each node processing. In this case, computation cost becomes a significant percentage of the overall response time. Therefore, PPLSA algorithm can deal with large datasets efficiently.

6 Conclusion

In this paper, we presented a parallel implementation of PLSA based on MapReduce. We use scaleup, sizeup and speedup to evaluate the performance. The experimental results show that it scales well through the machine cluster and has a nearly linear speedup. However, due to the limit memory, the algorithm do not work well when the dataset are too large. In the future, we will look into strategies to solve this problem.

Ackonwledgement. This work is supported by the National Natural Science Foundation of China (No. 60933004, 60975039, 61175052, 61035003, 61072085, 60903088), National High-tech R&D Program of China (863 Program) (No.2012AA011003).

References

1. Mei, Q., Zhai, C.: A note on EM algorithm for probabilistic latent semantic analysis. In: Proceedings of the International Conference on Information and Knowledge Management, CIKM (2001)
2. Steyvers, M.: Probabilistic Topic Models. In: Landauer, T., McNamara, D., Dennis, S., Kintsch, W. (eds.) Latent Semantic Analysis: A Road to Meaning. Laurence Erlbaum
3. Hofmann, T.: Probabilistic Latent Semantic Analysis. In: Proc. of 15th Conference on Uncertainty in Artificial Intelligence, pp. 289–296. Morgan Kaufmann, San Francisco (1999)
4. Hofmann, T.: Unsupervised learning by probabilistic latent semantic analysis. Machine Learning 42(1), 177–196 (2001)
5. Dean, J., Ghemawat, S.: MapReduce: Simplified Data Processing on Large Clusters. In: Proc. of Operating Systems Design and Implementation, San Francisco, CA, pp. 137–150 (2004)
6. Ghemawat, S., Gobioff, H., Leung, S.: The Google File System. In: Symposium on Operating Systems Principles, pp. 29–43 (2003)
7. Hadoop: Open source implementation of MapReduce (June 24, 2010), `http://hadoop.apache.org`
8. Han, J., Kamber, M.: Data Mining, Concepts and Techniques. Morgan Kaufmann (2001)
9. Lammel, R.: Google's MapReduce Programming Model - Revisited. Science of Computer Programming 70, 1–30 (2008)
10. Jin, Y., Gao, Y., Shi, Y., Shang, L., Wang, R., Yang, Y.: P^2LSA and P^2LSA+: Two Paralleled Probabilistic Latent Semantic Analysis Algorithms Based on the MapReduce Model. In: Yin, H., Wang, W., Rayward-Smith, V. (eds.) IDEAL 2011. LNCS, vol. 6936, pp. 385–393. Springer, Heidelberg (2011)