VotingRank: A Case Study of e-Commerce Recommender Application Using MapReduce

Jian-Ji Ren, Jae-Kee Lee
e-mail :jimey@donga.ac.kr
Department of Computer Engineering Dong-A University

Abstract

There is a growing need for ad-hoc analysis of extremely large data sets, especially at e-Commerce companies which depend on recommender application. Nowadays, as the number of e-Commerce web pages grow to a tremendous proportion; vertical recommender services can help customers to find what they need. Recommender application is one of the reasons for e-Commerce success in today's world. Compared with general e-Commerce recommender application, obviously, general e-Commerce recommender application's processing scope is greatly narrowed down. MapReduce is emerging as an important programming model for large-scale data-parallel applications such as web indexing, data mining, and scientific simulation. The objective of this paper is to explore MapReduce framework for the e-Commerce recommender application on major general and dedicated link analysis for e-Commerce recommender application, and thus the responding time has been decreased and the recommender application's accuracy has been improved.

1. Introduction

In recent years, recommender application has become one of the most popular tools to retrieve information from the web.

Since the number of e-Commerce web pages has also been increasing rapidly, it has become almost impossible to obtain information from the e-Commerce site without using recommender application. Users desire to define their preferences and customize the purchase information within the e-Commerce environment according to their individual needs. The site which does not have the recommender application services has to employ the search engines. The result from search engines is flat, and users have to go through the results to find what they want. It is time-consuming to locate their interesting products with the low relevance. In most situations, they are not able to evaluate all available alternatives and typically follow a two-step model to fulfill their purchasing processes. In the first step, they identify a subset of the available alternatives by choosing from a vast range of products, and, in a second step, they perform relative comparisons among these to arrive at their final decisions. Most people refer only to their interesting products. Therefore, it is imperative to make users browse their interesting products easier; employing the recommender application as a service.

Many recommender algorithms, such as collaborative filtering algorithms[1] and content-based filtering algorithms[2] have been designed for better recommender application. However, all these algorithms are based on general recommender application; which they cannot satisfy our users requirements.

The performance of intelligent recommender application is mainly evaluated by its accuracy and speed. The tolerable time of users waiting in front of the browser is generally limited, so the speed of the system responding to the users should be fast. On the other hand, users are usually anxious to receive the accurate information, so the accuracy of system should be high. The trends on data growth and processor speed improvement suggest that some form of parallelization is required to process the data. First, there are over 100.1 million websites operated as of March 2008[9]. The amount of data is growing at an exponential rate. In the e-Commerce world, the data is also growing at an amazing rate. Second, uniprocessor speed has stopped exponential growth since roughly 2002[13]. At the same time, traditional data processing algorithms promise to deliver efficient solutions to many of the problems arising from the interactions of consumers with the increasing volume of Internet applications. An efficient solution requires: 1) the distribution of the data to several computational nodes, 2) parallel accumulation of local data, and 3) aggregation of the results over all nodes. The researcher must manage the distribution to optimize bandwidth utilization, monitor the parallel accumulation of frequencies to handle straggling process and memory overruns, and synchronize the nodes and optimize bandwidth utilization for the aggregation of the result.

Recently, Jeffrey Dean and Sanjay Ghemawat in Google Corporation proposed a parallel framework, MapReduce[3], which automatically parallelizes and executes programs as users specified the appropriate map and reduce tasks, while other common work such as web indexing, data mining, and machine learning etc., are handled by the MapReduce system. The big volume of data that these internet application processes has led to interest in parallel processing on commodity clusters. The leading example is Google, which uses its MapReduce framework to process 20 petabytes of data per day[4].

In this paper, we propose a new parallel solution for the recommender application using MapReduce framework. Our approach has the following desirable properties:

Scalability: it implements that a MapReduce framework can abstract the complexity of running distributed data processing functions across multiple nodes in the cluster.

Efficiency: users could receive the accurate information over the algorithms.

Reliability: it implements that a MapReduce framework can divide data-parallel work to each node in the network.

The remainder of this paper is organized as follows. In section 2, we introduce background. In section 3, we give a detail description of our model. In section 4, we present our experimental results. Finally, we conclude the paper in section 5.

2. Background

MapReduce is a programming paradigm and framework developed by Google for simplified, parallel data processing on clusters of computers, while at the same time offering load balancing and fault tolerance. As its name implies, it was composed functions map and reduce. The origins of the map function in programming can be traced back to the LISP programming language[11] and reduce to APL[12], the precise specification being dependent on the implementation.
Dean and Ghemawat describe the MapReduce programming model as follow[3] :
The computation takes a set of *input* key/value pairs, and produces a set of *output* key/value pairs. The user of the MapReduce library expresses the computation as two functions: *Map* and *Reduce*.

Map, written by the user, 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

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.

MapReduce builds on the observation that many tasks have the same structure: a computation is applied over large data sets (e.g., documents) to generate partial results, which are then aggregated in some fashion. Taking inspiration from higher-order functions in functional programming language, MapReduce provides an abstraction that involves the programmer defining a “mapper” and a “reducer”, with the following signatures:

map: $(k_1, v_1) \rightarrow [(k_2, v_2)]$
reduce: $(k_2, [v_2]) \rightarrow [(k_3, v_3)]$

3. System model

If our aim is to study the e-Commerce recommender application over MapReduce programming, we must first the recommender application weighted rank theory.

3.1. VotingRank

Compared to Web page ranking, recommendation requires high weighted ranking the identification of production for consumers rather than generally popular products. As such, we propose a weighted ranking, named VotingRank, implementation of Bayes’ theorem.

$$VotingRank\ p_i = \sum_j G_{ji}\mu_j \qquad (1)$$

where G is the one item of the product’s weighted ranking, μ_j is the probability of G_{ji}, *and* $1 = \sum_1^n \mu_n$. The formula G is presented as follows:

$$G = \frac{Rn}{n+m} + \frac{Cm}{n+m} \qquad (2)$$

where:
R = voting for this item
n = number of votes for this item
m = minimum votes required to be recommended
C = average voting of this item across the whole data

Basically, the formula G is also presented as a matrix multiplication:

$$G = \frac{[R \quad C]\begin{bmatrix} n \\ m \end{bmatrix}}{n+m} \qquad (3)$$

3.2. Items of product

When a user surfs on the e-Commerce web site, she always focus on some items of product, which are the quality of product , the number of sale, the estimate of purchased by other consumers, etc. Based on estimate of those items, we can get the weighted ranking which is matrix representing by the VotingRank.

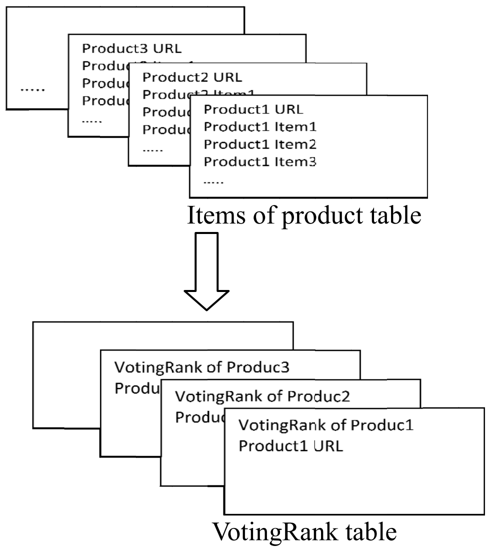

Fig.1. Items of product table and VotingRank table

3.3. MapReduce implementation

This presents the high-level requirements of what each MapReduce step of the program should do. MapReduce could be considered as two separate operators: Map and Reduce. In our current implementation, the Map function is arbitrary code that processes a set of input data and computes the results for a set of rows of the matrix and returns the (x, y) location of each item as the key and the result of the computation as the value. The reduce function computes these statistics across the entire data set in order to finally determine the high weighted ranking product.

```
//Map function
Map (k, v) {
        int  i = 0
             for each rating in data
                        i +=1
           rating += rating
              emit(Item, C)
}
```

Fig. 2. MapReduce for average voting for this item

```
//Map function
Map (k, v) {
             for each Item in data
                          emit(n, v)
}
//Reduce function
Reduce(k, v) {
             for each Item in data
           result = $\frac{[R \quad C]\begin{bmatrix} v \\ n \end{bmatrix}}{n+m}$
                     emit (p, result);
}
```

Fig. 3. MapReduce for VotingRank

4. Experiments

We have implemented the parallel construction and algorithm using JAVA programming language and the Hadoop middleware, an open-source implementation of MapReduce. Using the same code base allow us to compare the performance with MapReduce.

Many real datasets of user voting on different topic can be found on the Internet, including the IMDB movie ratings[5], the Jester joke ratings[6], the Book-Crossing book ratings[7] etc. Thus far, the GroupLens Research project has an ongoing MovieLens datasets[10], with ratings. To further investigate the MapReduce property, we applied our algorithms in two datasets of Movielens datasets. The first one consists of 100,000 ratings for 1682 movies by 943 users. The second one consists of approximately 1 million ratings for 3900 movies by 6040 users.

Distributing the task to a large cluster would clearly justify the overhead, but parallelizing to two or three machines would give virtually no benefit for the largest data set size we tested.

The experiments were conducted on a 15 node cluster (1 master, 14 slaves) with 2.0 GHz Intel Pentium 4 CPU, 512M RAM and 40GB 7200 RPM IDE hard disk space available per node. Every node was running Linux with kernel 2.6.24 and Hadoop 0.16.0[8]. All nodes were on the same 100Mbps Ethernet network.

Fig. 4 compares CPU time of processing two datasets. We observed that with processors (nodes) increasing, the CPU time of construction varies greatly when the datasets is larger. But the CPU time of construction falls down slowly for the small datasets because the workload per node is not full. Fig. 5 compares the speedup of processing two dataset with linear speedup.

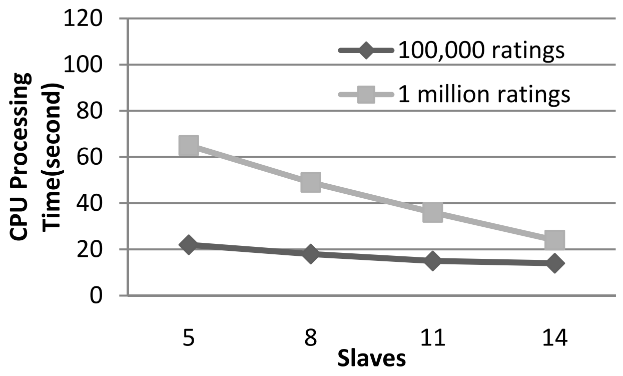

Fig. 4. CPU processing time

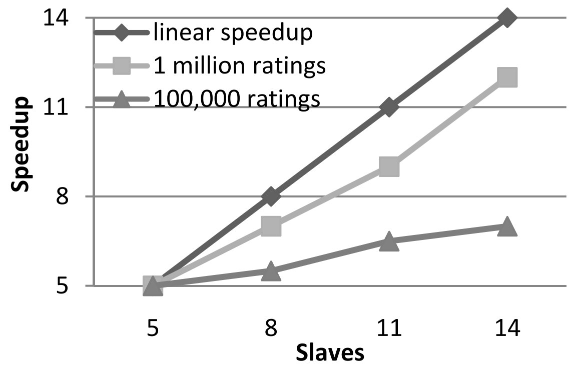

Fig. 5. Speedup

5. Conclusions

In this paper we have shown that an important class of model-building algorithms in recommender application can be straightforwardly recast into the MapReduce framework, yielding a distributed solution that is cost-effective, scalable, and reliable. Alternative strategies for parallelizing this algorithm either impose significant demands on the developer, the hardware infrastructure. They require making unwarranted independence assumptions, such as dividing the data sets into chunks and building separate models. We have further shown that on a 15-machine cluster of commodity hardware, the MapReduce implementations have excellent performance and scaling characteristics. It is our expectation that MapReduce will also provide solutions that the fast, easy, and cheap.

Overall, our work establishes that MapReduce provides a useful programming for recommender application.

References

[1] L.H.Ungar and D.P.Foster, Clustering Methods for Cllaborative Filtering, Proceedings of the Workshop on Recommendation Systems at the 15th National Conference on Artificial Intelligence, July 1998.

[2] M.Balabanovic and Y.Shoham, Fab: Conter-Based, Collaborative Recommendation, Communications of the ACM, Vol.40, No.3, pp.66-72, March 1997.

[3] J.Dean and S.Ghemawat. Mapreduce: Simplified data processing on large clusters. In OSDI'04: Sixth Symposium on Operating System Design and Implementation, December 2004.

[4] Jeffrey Dean and Sanjay Ghemawat . MapReduce: Simplified Data Processing on Large Clusters. In Communications of the ACM, Volume 51, Issue 1, pp. 107-113, 2008
[5] IMDB http://www.imdb.com/

[6] Jester joke ratings http://www.ieor.berkeley.edu/~goldberg/jester-data/

[7] Book-Crossing book ratings http://www.informatik.uni-freiburg.de/~cziegler/BX/

[8] Hadoop http://hadoop.apache.org/core/

[9] http://en.wikipedia.org/wiki/World_Wide_Web

[10] http://www.grouplens.org/node/73

[11] G.L.Steele Jr., Common Lisp: The Language, 2nd edition, Digital Press, 1990

[12] K.E. Iverson, A programming language. New York, NY, USA: John Wiley & Sons, Inc., 1962.

[13] D.Patterson and J.Henessy, Computer Architecture, A Quantitative Approach, Morgan Kaufmann Publishers, fourth edition, 2006.