A Parallelization Algorithm of Singular Value Decomposition

Jianfeng Duan1,a,* , Xian’e Cun2,b

1 Company of Xing Yu Shu Yun Of Chengdu, China
2 Dali University of Yunnan, China

*a Corresponding author e-mail: duanjianfeng@xysydata.com, b cxeyx@163.com

Abstract. Matrix factorization algorithm is one of the recommendable algorithms. In order to tackle the inefficiencies of the traditional matrix factorization algorithm like the slow training time and the insufficient computing resource for the mass data, a parallelization algorithm of singular value decomposition (SVD) under the Spark framework is proposed to perform SVD, standardization, and dimensionality reduction for the user-rating matrix, and obtain the user-feature matrix and project-feature matrix. The recommendation model is obtained by determining the prediction rating. MovieLens data show that this algorithm can significantly shorten the training time of the model, improve the running efficiency of the recommendation algorithms for the mass data, and improve the algorithm accuracy.

Keywords: Matrix factorization algorithm, recommendation algorithm, singular value decomposition, Spark, parallelization.

1. Introduction

The fast growth of the Internet has brought an explosive increase of the data size in the recent years, and the big data mainly come from the Internet service, among which personal data have exceeded the total amount of printing data as never before[1]. Against the backdrop of big data, how to capture the Internet users’ preference from mass data and offer them customized service has become a hot topic in the academia and the industrial circles[2]. Matrix factorization algorithm is one of the most popular algorithms among the collaborative filtering recommendation, and the project-based nearest neighbor algorithm estimates the project rating by the similarity between the projects[3]. However, inefficiencies of this algorithm such as cold start and the sparsity of data, in the context of fewer user rating, are significant. Literature[4] put forward an ALS-based matrix factorization algorithm which overcome the sparsity of data and is suitable to the fast parallel training. In early versions of Spark, the parallelization algorithm by this method is realized. Successors of Spark 1.1 have improved the ALS algorithm in MLlib into the ALS-WR algorithm[5], which makes the model parameters depend less on the size of the dataset, and easier to obtain the optimal parameters from the samples. But in the context of mass data, successive multiplication of the high-level matrix is required for the model training and the calculation complexity is high. During the competition of NetFlix Prize recommendation system Brandyn et al.[6] put forward the recommendation model of regularized matrix factorization(RMF) which has higher precision and better model effects. Literature[7] put forward an algorithm of singular value decomposition(SVD) which uses the method of stochastic gradient descent (SGD) and has an
advantage in precision and robustness. However, SGD would slow the convergence speed of the model and the iterative times are overmuch, which is limited in practical applications.

In the recommendation system adopting the matrix factorization algorithm, user-feedback information are converted into user-project matrix. However, in the context of a large number of users and mass data, the calculation time would become unserviceable and the calculation would be inefficient and even unavailable due to the limitation of memory in the stand-alone mode. Therefore, distributed algorithm has become a hot topic recently. Literature[8] put forward a MapReduce-based distributed algorithm of non-negative matrix factorization which can perform matrix decomposition for mass data. Literature[9] put forward an improved SGD decomposition algorithm and adopted the MapReduce framework for a better Speedup value[10]. But in the task execution of computational node, the MapReduce framework would generate many file reading which slow the efficiency of the algorithm.

For the sake of the realtime of the recommendation, an algorithm of singular value matrix decomposition, by virtue of the advantages of the Spark framework in terms of memory computing and parallelization calculation, is explored on the Spark platform to tackle the inefficiency of the matrix factorization algorithm for mass data.

2. Spark framework and singular value decomposition

2.1. Spark framework

Spark is an emerging engine for big data processing. It is developed by the UC Berkeley AMP lab and serves as one of the top open-source projects of Apache. Spark framework puts forward a cluster memory-based calculation model, expands the current MapReduce framework, and overcomes the low level of abstraction, inefficient expression, and poor real-time performance of this framework. Spark framework avoids repetitive disk operations of I/O by storing the cache into the memory and greatly improves the efficiency of data accessing.

Spark puts forward a distributed memory-abstract dataset - resilient distributed dataset(RDD)[11].RDD, as the main programming abstraction of Spark, is a user-defined set that is located in many computing nodes and performed via parallelization. When for transformations[12] by RDD, Spark can automatically generate directed acyclic graph(DAG). In case of fragmentation missing of one of the RDD, Spark can soon reconstruct according to the recorded DAG. At the same time, Spark would optimize DAG and improve the degree of parallelism of RDD operation. Even for complicated calculation on the disk, Spark is still more efficient than MapReduce. Besides, RDD can be saved in the memory or disk as per the user-defined rank, and directly process the stored data, saving much time for disk I/O and suitable to repetitive utilization of RDD.

Spark framework in the distributed circumstance consists of driver and executor, as shown in Fig. 1. Driver nodes undertake central coordination and regulation of every executor node. Executor nodes undertake calculation and every executor corresponds to an independent process and is parallel to other exectuors for calculation.

![Figure 1. Spark framework](image-url)
An executor is a process running on a worker node, responsible for the execution of certain tasks and for saving data to disk or memory. Task is a work unit sent to an executor, and the basic unit for running an application, and multiple tasks form a stage. An application often generates multiple jobs, and each job is split into multiple sets of tasks as a TaskSet, and the tasks in each TaskSet are of the same type with the name ‘stage’ which is divided by DAGScheduler. DAGScheduler constructs a stage-based DAG (Directed Acyclic Graph) on the job, and divides the stage based on the dependency between RDDs to find the scheduling method with the least overhead. The relationship between job, stage and task is shown in Figure 2.

![Figure 2. Spark computing unit](image)

### 2.2. Comparison between Spark and Hadoop

Hadoop is also a software platform under Apache to develop and run software for processing large-scale data. Distributed processing of large data sets on a number of computer clusters using a simple programming model is allowed. Hadoop emerged much earlier than Spark and Spark was designed to address the inefficiency of execution on the Hadoop platform. Core components of Hadoop include:

- **HDFS (Distributed File System):** solving massive data storage;
- **YARN (framework of job scheduling and cluster resource management):** solving the resource task scheduling;
- **MapReduce (programming framework of distributed computing):** solving the massive data computing.

The computational framework of Spark is extended based on the MapReduce framework. MapReduce provides a programming model through a simple abstraction of Mapper and Reducer, which allows concurrent and distributed processing of large data sets on an unreliable cluster of dozens or hundreds of PCs, while hiding the calculation details of concurrency, distribution (e.g., inter-machine communication), and failure recovery. The abstraction of Mapper and Reducer, in turn, is the basic element into which all kinds of complex data processing can be decomposed. In this way, complex data processing can be decomposed into a directed acyclic graph (DAG) consisting of multiple Jobs (containing a Mapper and a Reducer), and then each Mapper and Reducer can be placed on a Hadoop cluster and executed to produce results. The abstraction of Mapper and Reducer is the basic element that can be decomposed by all kinds of complex data processing. In this way, complex data processing can be decomposed into a directed acyclic graph (DAG) consisting of multiple Jobs (containing a Mapper and a Reducer), and then each Mapper and Reducer can be executed on a Hadoop cluster to generate results.

The MapReduce framework proposed by Hadoop abstracts the computation process and bring convenience to developers, but at the same time, drawbacks still exist:

- A Job has only two phases - Map and Reduce, and complex computation requires a large number of Jobs to complete, and the dependency between Jobs have to be managed by the developer;
- The intermediate results are also stored in the HDFS file system. The performance of iterative data processing is relatively poor;
• Reduce Task needs to wait until all the Map Task are completed before starting the computation;

• High latency, only suitable for batch data processing, not suitable for interactive data processing due to inadequate support of real-time data processing.

To solve the problems of Hadoop, Spark abstracts a new computational model and framework with the following main advantages:

• Spark not only improves the performance, but also provides a unified data processing platform for batch processing (Spark Core), interactive (Spark SQL), streaming (Spark Streaming), machine learning (MLlib), and graph computing (GraphX) through its framework.

• Spark enhances MapReduce to a higher level in the process of data processing by using Shuffle for much lower costs. By means of in-memory data storage and near real-time processing power, Spark performs many times faster than other technologies.

• Spark saves intermediate results in memory instead of writing to disk, which is a huge performance boost when the same data set needs to be processed many times.

• Spark supports more functions than Map and Reduce.

• Spark RDD is a high-level and abstract data collection for distributed big-data processing. Any operation on this collection can be as intuitive and easy as operating on an in-memory collection in functional programming, but operation on the is realized by a series of Tasks decomposed at the background and then sent to the cluster for completion.

Therefore, Hadoop MapReduce will be replaced by the new generation of big data processing platform as a technological trend, thereunto, Spark is the most widely recognized alternative.

2.3. Singular value decomposition

In the recommendation algorithm of matrix factorization, the original data is converted into a user-project matrix with m rows and n columns, denoted by R. R matrix is used as the input of the algorithm.

\[
R = \begin{bmatrix}
R_{11} & R_{12} & \cdots & R_{1n} \\
R_{21} & R_{22} & \cdots & R_{2n} \\
\vdots & \vdots & \ddots & \vdots \\
R_{m1} & R_{m2} & \cdots & R_{mn}
\end{bmatrix}
\]

Of which, m denotes the number of users, n denotes the number of project, \(R_{ij}\) denotes the interest value of the user i to the project j. For example, a film rating dataset denotes the score of the film by users, in which the rating range is from 1 to 5; 1 denotes dislike and 5 denotes most like.

Singular value decomposition of R is defined as follows:

\[
R = U \times \Sigma \times V^T
\]  

(1)

Of which, U is the orthogonal matrix of m*m, called left singular vector group, \(\Sigma\) is non-negative diagonal matrix of m*n with values on the diagonal as the singular values in the descending order; V is the orthogonal matrix of n*n, called right singular vector group. The matrix \(\Sigma\) is solely defined as:

\[
\Sigma = (\sigma_1, \sigma_2, \ldots, \sigma_m)
\]

(2)

Of which, the singular value \(\sigma_m = \sqrt{\lambda_m}\), \(\lambda_m\) denotes the feature value of the matrix \(RR^T\). Matrix U can be expressed as:

\[
U = (x_1, x_2, \ldots, x_m)
\]

(3)

Of which, \(x_m\) denotes the feature vector corresponding to the feature value \(\lambda_m\) of matrix \(RR^T\). Matrix V is determined via equation (1) by means of \(R^T\), U, and \(\Sigma^{-1}\).

\[
V = R^T U \Sigma^{-1}
\]

(4)

For the matrix of mass data, the whole original matrix does not have to be decomposed and only the first k singular values and corresponding singular vector groups are taken. This is because in many cases the singular values \(\sigma\) decrease fast, and the sum of the singular values of the first 10% or even the first 1% can occupy 99% of the sum of the whole singular values. Therefore, the singular values of the first k singular values is enough to describe the matrix.
The first k maximum singular values are obtained from $\Sigma$ as the diagonal element to form a new diagonal matrix $\Sigma_k$; the first k left and right singular vectors are obtained from U matrix and V matrix to form a new matrix $U_k$ and $V_k$, then $R_k$ is defined as:

$$R_k = U_k \times \Sigma_k \times V_k^T$$  \hspace{1cm} (5)

Then, $R_k$ is the optimal “k-rank matrix” similar to the matrix $R$ in terms of least square, that is, $R \approx R_k$.

3. Spark-based recommendation algorithm of parallelization

3.1. Singular value decomposition algorithm

User-project matrix can be decomposed into $R = U \times \Sigma \times V^T \approx U_k \times \Sigma_k \times V_k^T = U_k \Sigma_k^{1/2} (V_k \Sigma_k^{1/2})^T$. $U_k \Sigma_k^{1/2}$ can be used as user-feature matrix, and $V_k \Sigma_k^{1/2}$ can be used as project-feature matrix. In this way, $U_k \Sigma_k^{1/2}$ and $V_k \Sigma_k^{1/2}$ can be used to estimate the rating. The algorithm in literature[13] is improved into a recommendation algorithm of parallelization, with the steps shown as follows:

Step 1: The blank rating items in each row is filled with the average of the user rating from each row;
Step 2: Filled matrices are standardized, that is, the average of the user rating in one row is deducted from the elements in this row.
Step 3: Equation (5) is used to perform matrix decomposition and dimensionality reduction to obtain $U_k$, $V_k$, and $\Sigma_k$.
Step 4: The feature matrix $U_k \Sigma_k^{1/2}$ and $V_k \Sigma_k^{1/2}$ of users and projects are obtained;
Step 5: The estimated rating of project $i$ by the user $u - r_{ui} = U_k \Sigma_k^{1/2} (u) \times (V_k \Sigma_k^{1/2} (i))^T + \bar{r}_u$; that is, the $u$ row of $U_k \Sigma_k^{1/2}$ is multiplied by the $i$ row of $V_k \Sigma_k^{1/2}$, then added with the average of the rating by user $u$;
Step 6: TopN project is recommended to the user as per the estimated rating matrix.

3.2. Parallelization algorithm in the Spark framework

There exist excessive matrix multiplication operation in the process of recommendation algorithm of singular value decomposition, in which the algorithms between rows are independent, suitable to parallelization.

The algorithm is shown in detail as follows:
(1) input the training set to generate RDD[Rating]. The Rating entity include the property(uid,pid,rating).
(2) User-rating data is converted into sparse rating matrixRDD[Rating] $\map{\text{map}}$ Distributed RowMatrix.
(3) Fill the sparse matrix for standardization.
(4) Matrix decomposition is performed to generate k-rank left and right singular vector group and diagonal matrix of singular value, matrix $\map{\text{SVDfactorization}}$ svd.
(5) User-feature matrix and project-feature matrix are determined to obtain the estimated rating matrix. $\map{\text{multiply}}$ userMatrix, itemMatrix $\map{\text{multiply}}$ predictMatrix.

In the algorithm, (1)-(2) performs original data cleaning and processing, and convert the data into distributed user-project rating matrix, of which RDD[Rating] denotes the distributed dataset of the film rating and map denotes conversion; (4) decomposes the matrix into left and right singular vector group and diagonal matrix of singular value by means of the singular value decomposition algorithm, of which matrix is the distributed matrix obtained from (3), svd denotes left and right singular vector group and diagonal matrix of singular value, and SVDfactorization denotes the singular value decomposition algorithm; (5) determines the user-feature matrix via the left singular vector and singular value vector. Similarly, the project-feature matrix is determined, of which userMatrix denotes user matrix, and itemMatrix denotes feature matrix, predictMatrix denotes the estimation rating matrix, and multiply
denotes matrix multiplication. Finally, the predicted rating matrix is determined by the user-feature matrix and project-feature matrix.

4. Experiment and results

Three sets of physical machine are used, one is master and the other two are slave nodes with each one's memory of 8G, CPU of 2-core, and disk of 160G. Software environment: CentOs-6.6, Spark-1.5.2.

Scala language is used for the experiment, and IntelliJ is used for IDE, an HDFS is to save the data. MovieLens dataset of film rating are obtained from the website GroupLens(http://grouplens.org), and ml-100k dataset is used, including 10000 rating records of 1682 films made by 943 users. 100, 300, 500, 700, and 943 users are chosen from the dataset as samples and divided into a training set (80%) and a test set (20%), to obtained the operation time of algorithms at various node scale, as shown in Fig. 3.

![Fig.3 Comparison of the parallelization time](image)

Figure 3 shows that, for data sets of the same size, increasing the number of nodes can improve the running efficiency and shorten the runtime. As the data size increases, the increase of the runtime of the algorithm becomes slow. Unlike Hadoop's MapReduce which requires disk I/O operation for each task, Spark caches reused data in memory and improves the runtime efficiency, thus reflecting the advantages of Spark in terms of memory-based computing.

In addition, the improved operational efficiency from one to two nodes is lower than that from two to three nodes because the cluster contains at least one Master node, which not only performs computational tasks but is also responsible for the scheduling tasks of the cluster.

Speedup[10] is used to measure the parallelization performance of the algorithm with increased computational nodes at the same data size.

\[
\text{Speedup}_n = \frac{T_1}{T_n}; \quad n = 1, 2, \ldots \ldots
\]

of which, \( T_1 \) denotes the runtime of a single node; \( T_n \) denotes the runtime of \( n \) nodes. Ideally, the runtime of \( n \) nodes should be \( 1/n \) of the runtime of 1 node, i.e., Speedup coincides with the line \( y=x \).

Fig. 4 shows that the experimental Speedup values grow approximately linearly, and the larger the data volume, the faster the linear growth rate, indicating that the algorithm is suitable for matrix decomposition of mass data.
There are many metrics to evaluate the effectiveness of recommendation systems, which mainly consider the quality of the recommended results. For topN recommendations, two widely used metrics are used in the experiments: accuracy and recall [14]. In the experiments, the item set is divided into test set and topN set, and the item that appear in both test set and topN set are hit set. The accuracy and recall are defined as follows.

Accuracy:

$$\text{precision} = \frac{\text{size of hit set}}{\text{size of topN set}} = \frac{|\text{test} \cap \text{topN}|}{N}$$

Recall:

$$\text{recall} = \frac{\text{size of hit set}}{\text{size of test set}} = \frac{|\text{test} \cap \text{topN}|}{|\text{test}|}$$

In general, the two metrics tend to be negatively correlated; accuracy would decrease and recall would increase as the recommendation list increases. A two-dimensional metric containing both accuracy and recall can better reflect the performance of a recommendation system. Literature [15] proposed F1 metric, defined as follows:

$$F1 = \frac{2 \times \text{recall} \times \text{precision}}{\text{recall} + \text{precision}}$$

In the experiment, $N=10$, F1 value of each person is calculated and the average value is taken as the indicator of validation. The higher F1 value, the higher the accuracy of the algorithm. The improved singular value decomposition parallelization algorithm in this research is called DSVD (Distributed Singular Value Decomposition) algorithm, and the SVD algorithm in literature [13] is compared with this study, with the comparison results shown in the following figure.

Figure 5 shows that different degrees of dimensionality reduction have a significant impact on the accuracy of the algorithm. In addition, the DSVD algorithm proposed in this paper reveals a certain
improvement in recommendation accuracy compared with the traditional SVD algorithm, indicating that the algorithm proposed is suitable for recommendation for the massive data.

5. Conclusion
Matrix decomposition is one of the popular algorithms in recommendation algorithms in recent years. Based on previous research, the traditional SVD recommendation algorithm is improved into a parallelized SVD recommendation algorithm applicable to the Spark platform by virtue of the characteristics of matrix multiplication suitable for distributed operations and the advantages of memory computing-based Spark big data platform. Results show that the improved SVD algorithm not only brings higher acceleration ratio, but also improves the recommendation accuracy, which is suitable for recommendation for massive data. However, the algorithm also has shortcomings, and the task optimization of the algorithm was not performed in Spark environment for the test. In future, execution flows will be optimized in Spark environment to shorten the runtime.

References
[1] Ganz J, Reinsel D. The 2011 Digital Universe Study: Extracting value from Chaos[J]. IDC: Sponsored by EMC Corporation, 2011.
[2] MENG Xiangwu, JI Weiyu, ZHANG Yujie. A Survey of Recommendation Systems in Big Data [J]. Journal of Beijing University of Posts and Telecommunications, 2015, 38(2):1-15.
[3] Sarwar B, Karypis G, Konstan J, et al. Item-based collaborative filtering recommendation algorithms[C]//Proceedings of the 10th international conference on World Wide Web. ACM, 2001: 285-295.
[4] Hu Y, Koren Y, Volinsky C. Collaborative filtering for implicit feedback datasets[C]//Data Mining, 2008. ICDM'08. Eighth IEEE International Conference on. IEEE, 2008: 263-272.
[5] Zhou Y, Wilkinson D, Schreiber R, et al. Large-scale parallel collaborative filtering for the netflix prize[M]//Algorithmic Aspects in Information and Management. Springer Berlin Heidelberg, 2008: 337-348.
[6] Webb B. Netflix update: Try this at home[J]. Blog post sifter. org/simon/journal/20061211. html, 2006.
[7] Paterek A. Improving regularized singular value decomposition for collaborative filtering[C]//Proceedings of KDD cup and workshop. 2007, 2007: 5-8.
[8] Liu C, Yang H, Fan J, et al. Distributed nonnegative matrix factorization for web-scale dyadic data analysis on mapreduce[C]//Proceedings of the 19th international conference on World wide web. ACM, 2010: 681-690.
[9] Gemulla R, Nijkamp E, Haas P J, et al. Large-scale matrix factorization with distributed stochastic gradient descent[C]//Proceedings of the 17th ACM SIGKDD international conference on Knowledge discovery and data mining. ACM, 2011: 69-77.
[10] Zhang J, Li T, Ruan D, et al. A parallel method for computing rough set approximations[J]. Information Sciences, 2012, 194: 209-223.
[11] Zaharia M, Chowdhury M, Das T, et al. Resilient distributed datasets: A fault-tolerant abstraction for in-memory cluster computing[C]//Proceedings of the 9th USENIX conference on Networked Systems Design and Implementation. USENIX Association, 2012: 2-2.
[12] Karau H, Konwinski A, Wendell P, et al. Learning Spark: Lightning-Fast Big Data Analysis[M]. " O'Reilly Media, Inc.", 2015.
[13] Sarwar B, Karypis G, Konstan J, et al. Application of dimensionality reduction in recommender system-a case study[R]. Minnesota Univ Minneapolis Dept of Computer Science, 2000.
[14] ZHU Yuxiao, LÜ Linyuan. Evaluation Metrics for Recommender Systems [J]. Journal of University of Electronic Science and Technology of China, 2012, 41(2): 163-175.
[15] Pazzani M, Billsus D. Learning and revising user profiles: The identification of interesting web sites[J]. Machine learning, 1997, 27(3): 313-331.