Hindawi
Scientific Programming
Volume 2021, Article ID 8841133, 15 pages
https://doi.org/10.1155/2021/8841133

Research Article

Collaborative Filtering Recommendation Using Nonnegative Matrix Factorization in GPU-Accelerated Spark Platform

Bing Tang,[1] **Linyao Kang**,[1] **Li Zhang**,[1] **Feiyan Guo**,[1] **and Haiwu He**[2]

[1]*School of Computer Science and Engineering, Hunan University of Science and Technology, Xiangtan 411201, China*
[2]*Shandong Computer Science Center (National Supercomputer Center in Jinan), Jinan 250014, China*

Correspondence should be addressed to Bing Tang; btang@hnust.edu.cn

Received 1 October 2020; Revised 16 December 2020; Accepted 21 December 2020; Published 5 January 2021

Academic Editor: Shah Nazir

Nonnegative matrix factorization (NMF) has been introduced as an efficient way to reduce the complexity of data compression and its capability of extracting highly interpretable parts from data sets, and it has also been applied to various fields, such as recommendations, image analysis, and text clustering. However, as the size of the matrix increases, the processing speed of nonnegative matrix factorization is very slow. To solve this problem, this paper proposes a parallel algorithm based on GPU for NMF in Spark platform, which makes full use of the advantages of in-memory computation mode and GPU acceleration. The new GPU-accelerated NMF on Spark platform is evaluated in a 4-node Spark heterogeneous cluster using Google Compute Engine by configuring each node a NVIDIA K80 CUDA device, and experimental results indicate that it is competitive in terms of computational time against the existing solutions on a variety of matrix orders. Furthermore, a GPU-accelerated NMF-based parallel collaborative filtering (CF) algorithm is also proposed, utilizing the advantages of data dimensionality reduction and feature extraction of NMF, as well as the multicore parallel computing mode of CUDA. Using real MovieLens data sets, experimental results have shown that the parallelization of NMF-based collaborative filtering on Spark platform effectively outperforms traditional user-based and item-based CF with a higher processing speed and higher recommendation accuracy.

1. Introduction

In recent years, the scale of data has grown exponentially. Globally, it is becoming a trend to research and develop big data technology, use big data to promote economic development, improve social governance, and improve government services and regulatory capabilities. How to effectively extract knowledge from big data, understand and analyze it, and finally make predictions are current popular research topics.

As an important mathematical tool for big data processing, nonnegative matrix factorization is a matrix decomposition approach that decomposes a nonnegative matrix into two low-rank matrices constrained to have nonnegative elements [1, 2]. This results in a reduced representation of the original data that can be seen either as a feature extraction or as a dimensionality reduction technique. The widespread usage of the NMF is due to its ability of providing new insights and relevant information about the complex latent relationships in experimental data sets. Since Lee and Seung's Nature paper [1], NMF has been extensively studied and has a great deal of applications in science and engineering. It has become an important mathematical method in machine learning and data mining and has been widely used in feature extraction, image analysis [3], audio processing [4], recommendation systems [5, 6], pattern recognition, data clustering [7], topic modeling [8], text mining [9], bioinformatics [10], and so forth. Unlike other factorization methods (e.g., PCA, ICA, SVD, VQ, etc.), NMF can be interpreted as a parts-based representation of the data because only additive combinations are allowed. In contrast to PCA and ICA, NMF strictly requires that the entries of both resulting matrices be nonnegative. Such a constraint is very meaningful in many applications, in which the data representation is purely additive; for instance, the user ratings of e-commerce websites are usually nonnegative values, and image pixels are nonnegative values.

The main problem of NMF is that the original matrix is usually high-order matrix, which makes the computational

complexity very high. Therefore, the parallel algorithm of NMF gradually attracts more attention, and some parallel NMF algorithms have been proposed. Although the parallelization of NMF can improve the computational efficiency to a certain extent, parallel algorithms should be matched to the machine hardware architecture and should have strong scalability, that is, the ability to effectively utilize increased processor resources.

Accelerating HPC applications is currently under extensive research using new hardware technologies such as the recent Central Processing Units (CPUs) that are getting multiple processor cores for parallel computing, Graphics Processing Units (GPUs) that process huge data blocks in parallel, and hybrid CPUs/GPUs computing which is a very common solution for HPC. GPUs are getting more attention than other HPC accelerators due to their high computation power, strong performance, functionality, and low price. The modern GPU is not only a powerful graphic engine but also a highly parallel programmable processor featuring peak arithmetic and memory bandwidth [11]. They are now used to accelerate graphics and some general applications with high data parallelism (GPGPU) due to the availability of Application Programming Interfaces (APIs), such as Compute Unified Device Architecture (CUDA) and Open Computing Language (OpenCL).

Spark is a distributed in-memory computation framework that was proposed by AMPLab of University of California, Berkeley, in 2009 and is based on a framework of processing large amounts of data in memory [12, 13]. It supports four programming languages, Scala, Java, Python, and R. Resilient Distributed Datasets (RDD) is a new concept proposed by Spark for data collections. RDD can support coarse-grained write operations [14]. Spark caches a particular RDD into memory, and the next operation can read directly from memory. The data is not written to disk, saving a lot of disk I/O overhead. Experimental performance evaluation confirmed that Spark's performance has increased by dozens or even 100 times compared to Hadoop, which relies on MapReduce model [15, 16] and data being stored in a distributed file system called HDFS rather than in memory.

Currently, some parallel approaches for nonnegative matrix factorization have been proposed, for example, high-performance approaches using message passing interface (MPI) [17], GPU-accelerated approaches [9, 18], and Hadoop-based MapReduce approaches [10, 19]. These approaches mainly utilize the multicore characteristics of the system, and there is still the potential to improve performance by utilizing memory, CPU, and GPU resources together.

Meanwhile, collaborative filtering is a method of making automatic predictions (filtering) about the interests of a user by collecting preferences from group users (collaborating). The underlying assumption of the collaborative filtering approach is that if a person A has the same opinion as a person B on an issue, A is more likely to have B's opinion on a different issue than that of a randomly chosen person. Through calculating the data similarity between two users, we can get the similarity between two users. Although traditional collaborative filtering is extremely successful in the recommendation system, as the data increases, the recommendation algorithms have been confronted with various problems, such as scalability problems, cold start problems, and matrix sparseness problems.

In order to solve the above problems, this paper proposes a combination of Spark-based and GPU-based acceleration model to develop scalable NMF parallel algorithm, which takes advantages of both GPU and in-memory computing, to obtain a highly scalable parallel NMF algorithm. Furthermore, this paper proposes a collaborative filtering method based on NMF and is parallelized and migrated to the Spark platform equipped with GPU to execute, which effectively improves the calculation efficiency, and thus can update the recommendation results faster and produce more accurate recommendation results. The experimental results show that the parallelization of NMF-based collaborative filtering effectively improves the calculation efficiency and accuracy.

The rest of the paper is organized as follows. Section 2 surveys related work. Section 3 introduces the mathematical fundamental of NMF and describes the general parallel principle of NMF. Section 4 describes the architecture of GPU-accelerated Spark platform and presents GPU-accelerated NMF on Spark. Section 5 introduces the collaborative filtering algorithm based on NMF. Section 6 presents performance evaluation results of GPU-accelerated NMF on Spark and collaborative filtering-based NMF, which is followed by conclusions in Section 7.

2. Related Work

2.1. Hybrid Big Data Processing Model. Due to the diversity of hardware equipment in high-performance computing system, in order to deal with the real-world complex applications, the mix of different computing modes has become a major direction, such as hybrid CPU/GPU and CPU/FPGA. From the model point of view, there are hybrid MapReduce/CUDA model [20], hybrid MapReduce/MIC model [21], hybrid OpenMP/MPI model [22], and so forth. In recent years, the practical experience of academia and industry has shown that computing platforms based on heterogeneous CPU/GPU system have great development potential and have attracted more and more attentions [11, 23].

2.2. Parallel Nonnegative Matrix Factorization. To handle large data sets through nonnegative matrix factorization, there are three main directions. The first class of algorithms is called online NMF algorithms [6, 24, 25], which are the oldest approach for dealing with high-dimensional data processing through NMF. The second class is known as distributed NMF algorithms, which distribute data over a network so that several small-scale data can be performed concurrently. The third class of algorithms is called compressed NMF algorithms [26, 27], which perform structured random compression to project the data onto the lower-dimensional manifolds. In this paper, we only focus on distributed NMF.

Nonnegative matrix factorization is usually solved through alternate iteration [2], which makes it suitable for parallelization. Three aspects that restrict the scalability of the parallel NMF algorithm are listed as follows: synchronization between computing processes, data loading and data transmission, and parallel granularity division. At present, some parallelization algorithms have been proposed to accelerate nonnegative matrix factorization.

Janecek et al. used linear algebra toolkits such as BLAS, LAPACK, and ARPACK to implement multithreaded programs on a single computer to perform efficient NMF [28]. Lopes and Ribeiro implemented a GPU-based machine learning library named GPUMLib, which contains an implementation of GPU-accelerated NMF [29]. Kysenko et al. also applied GPU-accelerated NMF to text mining [9]. Battenberg and Wessel implemented a parallel NMF, using the characteristics of a shared memory multicore system based on OpenMP and a many-core GPU based on CUDA technology, and applied it to audio signal processing, but it can only work for a single node [30]. A parallel NMF based on the combination of MPI and GPU is implemented in [18], and it is used for biological sequence comparison. Tang et al. proposed a hybrid parallel hierarchical NMF algorithm based on OpenMP and MPI [31].

Using some high-performance computing software packages, such as ParMETIS, ScaLAPACK, and HPSEPS, we can develop nonnegative matrix factorization parallel algorithms based on MPI/OpenMP/GPU using these software packages, which are ultimately not suitable for practical big data processing in Internet era. On the basis of open-source big data processing framework such as Hadoop and Spark, it is a more suitable idea to develop a parallel algorithm of NMF to make it suitable for Internet big data processing. In [10], Liao et al. realized the distributed NMF based on MapReduce for biological data processing. Sun et al. realized large-scale NMF based on MapReduce in [32], and Liu et al. also proposed a distributed NMF based on MapReduce for processing large-scale web data using Hadoop streaming method [19]. In our previous work [33], we proposed a parallel NMF algorithm in Spark platform, which makes full use of the advantages of in-memory computation mode.

2.3. Parallel and Distributed Collaborative Filtering. Many e-commerce companies have already incorporated recommendation systems with their services, for example, product recommendations by Amazon (http://www.amazon.com) and Taobao (http://www.taobao.com) and movie recommendations by Netflix (http://www.netflix.com). The implementations and algorithms of collaborative filtering for the applications of recommendation systems face several challenges. First is the size of processed datasets. The second one comes from the sparseness of rating matrix, which means for each user only a relatively small number of items are rated. With the increase of a large amount of data and the complexity of the data, it is confronted with the problem of low efficiency. Thus, highly efficient collaborative filtering algorithm is needed.

Recently, great interest has turned towards parallel and distributed implementations of collaborative filtering algorithms. They can be classified into several categories: (1) distributed memory implementations, such as the MPI-based parallel implementations proposed in [34, 35]; (2) shared memory implementations, such as MPI implementations in [36, 37]; (3) GPU-based implementations, such as [38, 39]; (4) platform-based implementations, such as the Hadoop platform-based collaborative filtering implemented by [40, 41]; and (5) heterogeneous implementation, such as the hybrid OpenMP + MPI implementation proposed by Narang et al. [42] and Karydi and Margaritis [43].

On the other hand, these challenges of collaborative filtering have been well taken care of by matrix factorization (MF). Matrix factorization methods have recently received greater exposure as unsupervised learning methods for latent variable decomposition and dimensionality reduction [44]. It is a powerful technique to find the hidden structure behind the data. There are several matrix factorization models that could be used for collaborative filtering recommendations, for example, Singular Value Decomposition (SVD) [45, 46], Principal Component Analysis (PCA) [47], Probabilistic Matrix Factorization (PMF) [48], and nonnegative matrix factorization [49]. However, efficient collaborative filtering algorithm also places demands on fast matrix factorization.

To sum up, high-dimensional NMF is time-consuming, and there is an urgent need for high-performance parallelization solutions. At present, there is no flexible distributed processing framework that considers both the memory computing mode and GPU technologies for NMF at the same time. Considering Spark distributed processing framework and combining the powerful computing advantages of GPU and large-capacity memory, large-scale NMF parallel algorithm would enable the algorithm to be easily adapted to the processing of Internet big data. To the best of our knowledge, it is the first NMF-based collaborative filtering implementation that is parallelized and migrated to the Spark platform equipped with GPU. Experimental results validate that the parallelization of NMF-based collaborative filtering on Spark platform effectively improves the calculation efficiency and accuracy.

3. Parallel Nonnegative Matrix Factorization

3.1. Nonnegative Matrix Factorization. Nonnegative matrix factorization seeks to approximate a nonnegative $n \times m$ matrix V (in this context, a matrix is called nonnegative if all of its elements are nonnegative) by a product $V \approx WH$ of nonnegative matrices W and H of dimensions $n \times r$ and $r \times m$, respectively, with a given and typically low maximal rank r. Usually, r is chosen to satisfy $r = \min\{m, n\}$ such that WH can be thought of as a compressed form of the original data. It forms the basis of unsupervised learning and data reduction algorithms with applications to image recognition, speech recognition, data mining and collaborative filtering, and so forth.

NMF is able to represent a large input data set as the linear combination of a reduced collection of elements named *factors*. In this way, W contains the reduced set of r factors, and H stores the coefficient of the linear combination of such factors which rebuilds V. NMF iteratively modifies W and H until their product approximates to V. Such modifications, composed by matrix products and other algebraic operations, are derived from minimizing a cost function that describes the distance between WH and V. Lee and Seung presented two NMF algorithms based on multiplicative update rules whose objective functions are *Square of Euclidean Distance* (SED) and *Generalized Kullback-Leibler Divergence* (GKLD), respectively [1, 2]:

$$E(V \| WtH) = \frac{1}{2} \sum_{i=1}^{m} \sum_{j=1}^{n} \left(v_{ij} - \sum_{k=1}^{r} w_{ik} h_{kj} \right)^2, \quad (1)$$

$$D(V \| WtH) = \sum_{i,j} \left(v_{ij} \log \frac{v_{ij}}{(WH)_{ij}} - v_{ij} + (WH)_{ij} \right). \quad (2)$$

Then, the objective of NMF is converted to optimize the following: $\min_{W,H} E(V \| WtH)$ or $\min_{W,H} D(V \| WtH)$, and s.t. $W, H \geq 0$, $\sum_{i=1}^{n} w_{ij} = 1, 1 \leq j \leq r$.

In this paper, we define SED as the objective function, so we have $\min(\|V - WH\|_F^2)$, which leads to the updating rules for matrices H and W:

$$h_{ij} = h_{ij} \frac{\left(W^T V\right)_{ij}}{\left(W^T WH\right)_{ij}}, \quad (3)$$

$$w_{ij} = w_{ij} \frac{\left(VH^T\right)_{ij}}{\left(WHH^T\right)_{ij}}. \quad (4)$$

3.2. Parallel Nonnegative Matrix Factorization. Before describing our experimental study, we briefly introduce the main existing parallel techniques of NMF. By analyzing equations (1) and (2), we can get the basic principle of iteration calculation of NMF in parallel manner. Matrix operations are performed in blocks. The block-based parallel updating rules for matrices H and W over multiple processes are shown in Figure 1, and the size of b_m can be adjusted according to the hardware configurations. At the time of initialization, initial W and H are produced. Because SVD-based initialization has been proven to be more effective for iteration of H and W [50], we generate initial W and H by the method of SVD. As you see, the size of matrix W is $n \times r$, the size of the matrix block V_j is $n \times b_m$, and the size of the matrix block H_j is $r \times b_m$, and finally the updated matrix block H_j is obtained. As shown in Figure 1(b), the new matrix H is used to compute the new matrix block W_i and so on. Matrix H and W are updated alternatively.

It can be seen from the analysis that the original matrix V is equivalent to a read-only variable, which is shared among all processes. With the iteration, matrices W and H need to be synchronized among all processes. The algorithm works by iteratively all-gathering the entire matrix H or W to each processor and then performing the Local Update Computations to update W_i or H_j.

4. GPU-Accelerated NMF on Spark

4.1. Spark. Conceptually, Apache Spark is an open-source in-memory data analytics cluster computing framework [12, 13]. As a MapReduce-like cluster computing engine, Spark also possesses good characteristics such as scalability and fault tolerance as MapReduce does. The main abstraction of Spark is RDDs, which make Spark be well qualified to process iterative jobs, including PageRank algorithm and K-means algorithm. RDDs are unique to Spark and thus differentiate Spark from conventional MapReduce engines. In addition, on the basis of RDDs, applications on Spark can keep data in memory across queries and reconstruct automatically data lost during failures. RDD is a read-only data collection, which can be either a file stored in an external storage system, such as HDFS, or a derived data set generated by other RDDs. RDDs store much information, such as its partitions, and a set of dependencies on parent RDDs called lineage. With the help of the lineage, Spark recovers the lost data quickly and effectively. Spark shows great performance in processing iterative computation because it can reuse intermediate results and keep data in memory across multiple parallel operations.

4.2. GPU-Accelerated Spark Platform. Modern GPUs are now capable of general computing. Due to the popularity of the CUDA on Nvidia GPUs, which can be considered as a C/C++ extension, we will mostly follow CUDA terminologies to introduce GPU computing. Current generations of GPUs are used as accelerators of CPUs and data are transferred between CPUs and GPUs through PCI-E buses. NVIDIA GPU programming is generally supported by the NVIDIA CUDA environment. A program on the host (CPU) can call a GPU to execute CUDA functions called kernel.

GPU is a multicore processor designed to parallelizable computational intensive tasks. It has very high computational processing power and data throughput. In scientific research and practical applications, the parallelizable computing task modules with less logical processing in the system are often transplanted to the GPU for execution, and a large execution performance improvement can usually be achieved.

However, Spark cluster will slow down when processing extremely large-scale data sets, especially when the node number is not very high. At the same time, more and more developers use GPUs for parallel computing to obtain high throughput and performance. Combining Spark with GPU, the mixed architecture is quickly becoming an emerging technology, which embeds the GPU into Spark, implements CPU/GPU integration, and builds an efficient heterogeneous parallel system.

In the CPU/GPU heterogeneous parallel cluster, the CUDA-based GPU acceleration technology is used, and the Spark computing tasks are accelerated by GPU. The basic

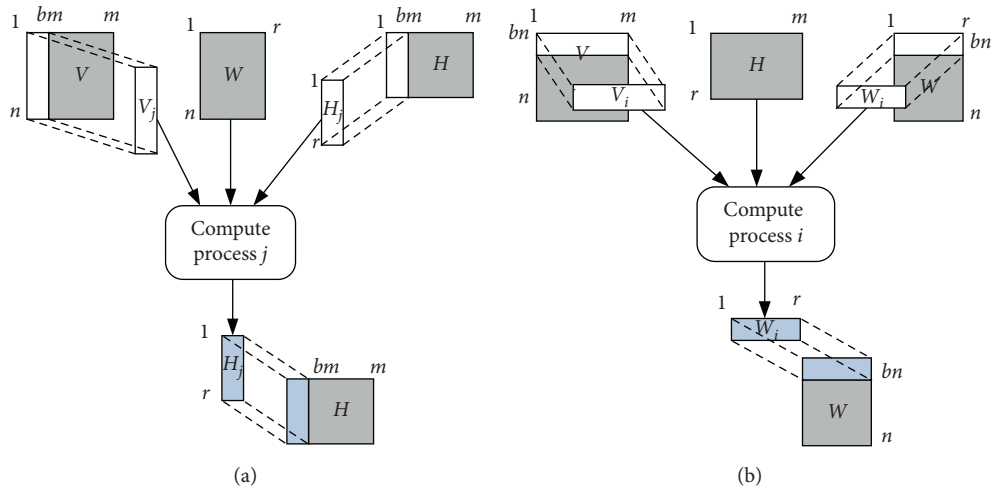

Figure 1: Block-based parallel updating rules for matrices H and W over multiple processes. (a) Iteration of matrix H and (b) iteration of matrix W.

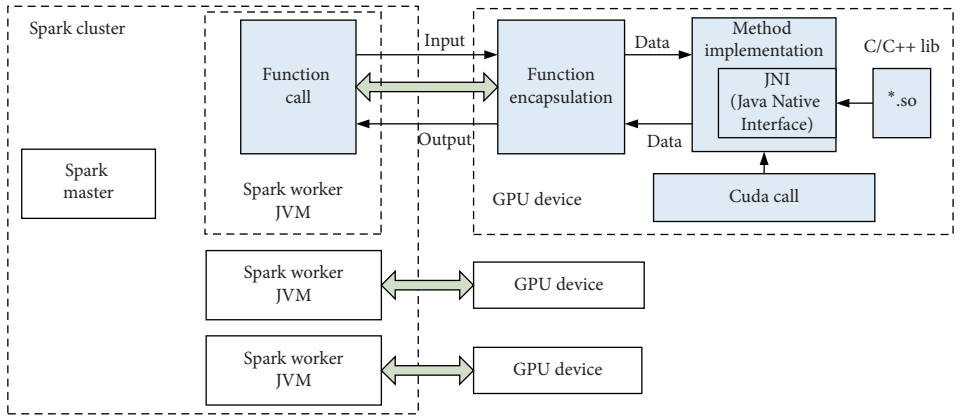

Figure 2: Architecture of GPU-accelerated Spark platform.

idea is that a part of operations of the Spark RDD are transferred to the GPU cores. GPU code execution flows are as follows: (1) copy data from main memory to GPU global memory; (2) GPU is driven by CPU instructions; (3) GPU parallel processing is in each core; and (4) GPU returns results to main memory. According to this idea and combined with Spark workflow, the GPU code is encapsulated, and then the data is transmitted between Spark Worker and GPU. The basic principle of Spark-GPU fusion is shown in Figure 2.

From the perspective of programming language, since the GPU program is usually developed in C/C++ language and the Spark platform uses Java language for program develop, Java's JNI (Java Native Interface) technology provides a solution to bridge the GPU and Spark through code encapsulation to implement interfaces for the Worker to call. Several JNI tools for GPU programming can be used. For example, JCuda (http://www.jcuda.org) is a development kit that provides bindings to the CUDA runtime, which currently includes multiple packages such as JCublas, JCufft, JCurand, JCusparse, JCusolver, JCudpp, JNpp, and JCudnn etc. It is convenient to write GPU programs in Java language. Other user-defined GPU programs written in C/C++ can also be called after being packaged into Java functions.

For the developers, a bidirectional transmission channel between the main memory and the GPU global memory should be established. If the operation of the RDD is transferred to the GPU core, high-speed data transmission between the main memory and the GPU global memory is required, which is also implemented by function encapsulations, as is demonstrated in Figure 2.

4.3. GPU-Accelerated NMF. As we demonstrated the matrix iterative process in equations (3) and (4) and Figure 1, the main principle of GPU-based parallel NMF is presented in Figure 3. The basic idea of GPU-based parallel NMF is to design several kernel functions to implement update rules for matrices H and W. H and W are blockwise-transferred. In Figure 3, circled operations denote CUDA kernels, and "$\cdot*$" and "$\cdot/$" denote point-wise matrix operations, multiplication, and division, respectively. Most of the matrix operations can be implemented using the libraries of Cublas and Cusparse, together with two self-defined operations, dot multiplication and dot division. In order to reduce the

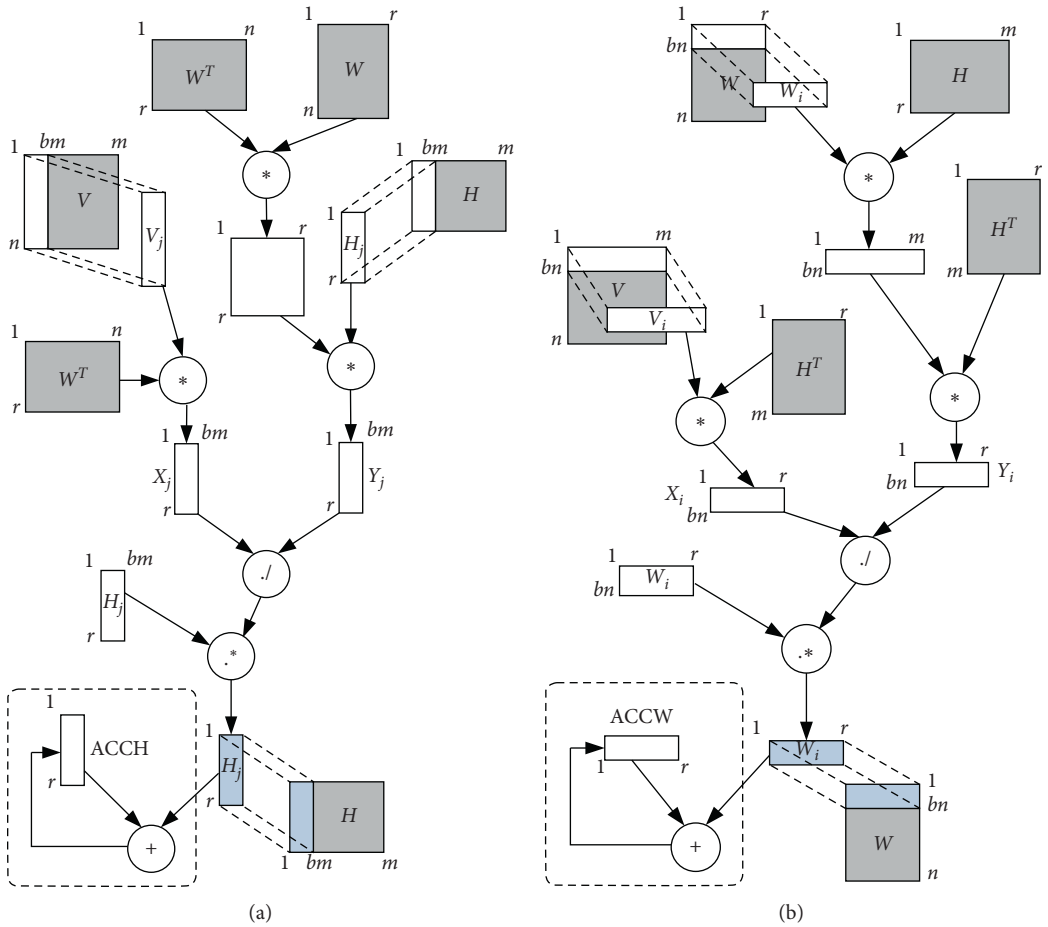

Figure 3: GPU implementation of iteration.

programming difficulty, JNI technology is used to transfer the CUDA programs to Java function encapsulations, which are called by Spark executors.

4.4. GPU-Accelerated NMF on Spark. Spark has advantages in iterative computing, and GPU has advantages in numerical calculation of vectors and matrices. In the Spark-GPU fusion platform, fast memory read and write, combined with GPU acceleration, can play their respective advantages to improve performance. NMF calculation is started and controlled by Spark driver. The Workers calculate the parallel tasks iteratively in a distributed manner. Workers are optimized with the highest speed using the GPU device and running the GPU kernel functions to complete the task. All intermediate results are written to the memory in each iteration and exchanged among the Workers and sent to the GPU global memory. Until the iterations are terminated, the tasks are completed and the results are written to HDFS.

The whole algorithm is described in Algorithm 1. Matrix V is broadcasted to all executors, and each Worker obtains the corresponding matrix block W_i or H_j from RDD. In the Spark platform, after the Action operator is triggered, all accumulated operators form a directed acyclic graph. Task is split into different stages based on different dependencies between RDDs. One stage consists of a series of function execution pipelines. The stages of GPU-accelerated NMF through RDD are listed as follows:

Stage 1: Read and convert matrices W and H; perform *mapPartition* function to update H blocks;

Stage 2: Splice all blocks of H after one iteration through performing a *collect* operation;

Stage 3: Read and convert matrices W and H; perform *mapPartition* function to update W blocks;

Stage 4: Splice all blocks of W after one iteration through performing a *collect* operation, and prepare for the next iteration.

Then, iteratively preform the above four stages. The method of caching data in memory is much faster than in file system for each iteration. When the convergence condition is reached, the matrices updating is terminated, and the results are then written to HDFS.

Input: Original matrix $V_{n\times m}$, low rank r and iteration times *iter*
Input: Context of Spark Environment *sc*
Input: Number of executors *en* and number of data partitions *pn*
Input: Data collection of matrix elements *dc M* and *dc H* for matrices M and H
Input: Data collection in the form of RDD *rdd M* and *rdd H* for matrices M and H
Output: Matrices $W_{n\times r}$ and $H_{r\times m}$ after decomposition
(1) generate initial W, H by random
(2) $dc\,W \leftarrow W, dc\,H \leftarrow H$
(3) broadcast V
(4) **for** $k = 1$: iter **do**
(5) $rdd\,H \leftarrow sc.parallelize(dc\,H, pn)$
(6) //update H
(7) call $rdd\,H.mapPartition(mapH)$
(8) **function** mapH(data, result)
(9) $X \leftarrow gpu\_multiply(W^T, V)$
(10) $WW \leftarrow gpu\_multiply(W^T, W)$
(11) $Y \leftarrow gpu\_multiply(WW, data)$
(12) $data \leftarrow gpu\_dot\_multiply(data, X)$
(13) $result \leftarrow gpu\_dot\_divide(data, Y)$
(14) return result
(15) **end function**
(16) $dc\,H \leftarrow rdd\,H.collect$
(17) $rdd\,W \leftarrow sc.parallelize(dc\,W, pn)$
(18) //update W
(19) call $rdd\,W.mapPartition(mapW)$
(20) **function** *mapW(data, result)*
(21) $X \leftarrow gpu\_multiply(V, H^T)$
(22) $WH \leftarrow gpu\_multiply(W, H)$
(23) $Y \leftarrow gpu\_multiply(WH, H^T)$
(24) $data \leftarrow gpu\_dot\_multiply(data, X)$
(25) $result \leftarrow gpu\_dot\_divide(data, Y)$
(26) $return\,result$
(27) **end function**
(28) $dc\,W \leftarrow rdd\,W.collect$
(29) **end for**
(30) $W \leftarrow dc\,W, H \leftarrow dc\,H$

Algorithm 1: GPU-accelerated NMF on Spark.

5. Collaborative Filtering Algorithm Based on NMF

5.1. Classic Collaborative Filtering Algorithm. Collaborative filtering recommendation algorithms can be divided into two categories: user-based CF and item-based CF. The recommendation process based on collaborative filtering can be described as three stages:

Stage 1: *Collect user preferences.* After preprocessing the user behavior data, according to different behavior analysis methods, you can choose grouping or weighting to obtain a "user-item" preference matrix V whose size is $n \times m$, where n is the number of users, m is the number of items, and matrix element v_{ij} denotes the i-th user's preference for the j-th item, which is generally a floating point number in the range [1, 5] or a binary value of 0 or 1. The value highly depends on the content of the item. If the item is a commodity in e-commerce, the value indicates whether the user purchased or not. Sometimes, it means whether the user watched or not, or the interest is like or dislike, or the interest is high or low.

Stage 2: *Discovery of similar users or items.* In the "user-item" preference matrix, a user's preference for all items is used as a vector to calculate the similarity between users to obtain the similarity matrix sim. For a specific user u, from the remaining $n - 1$ users in the system, the similarity value corresponding to the user u is sorted in descending order; the k-nearest neighbor users with the largest similarity value are selected to form the nearest neighbor user set $N = \{n_1, n_2, \ldots, n_k\}$.

For the item-based CF, all users' preferences for an item are regarded as a vector to calculate the similarity between items. Generally, there are three common methods for calculating similarity: Euclidean distance, Pearson correlation coefficient, and Cosine similarity. This paper uses Pearson correlation coefficient as an example [44]. The reason why we choose Pearson correlation coefficient is that, different from the Euclidean distance, Pearson correlation coefficient is able

to reduce the grade inflation error, which is relevant in the recommendation domain. The formula for calculating the similarity using Pearson correlation coefficient is presented as follows:

$$\mathrm{sim}(u_1, u_2) = \frac{m\sum_{j=1}^{m} v_{u_1 j} v_{u_2 j} - \sum_{j=1}^{m} v_{u_1 j} \sum_{j=1}^{m} v_{u_2 j}}{\sqrt{m\sum_{j=1}^{m} v_{u_1 j}^2 - \left(\sum_{j=1}^{m} v_{u_1 j}\right)^2} \sqrt{m\sum_{j=1}^{m} v_{u_2 j}^2 - \left(\sum_{j=1}^{m} v_{u_2 j}\right)^2}}, \tag{5}$$

where u_1 and u_2 denote two users, $\mathrm{sim}(u_1, u_2)$ is the similarity of users u_1 and u_2, and v_{u1j} and v_{u2j} are the ratings of j-th items given by users u_1 and u_2.

Stage 3: *Generate the prediction matrix and Top-N recommendation results.* Using the score given by the nearest neighbor on the item, the user's score on the specific item is calculated through the weighted average of the similarities. Suppose that user u's nearest neighbor set $N = \{n_1, n_2, \ldots, n_k\}$; user u's prediction score for an item i is denoted as v'_{ui}, which is shown in the following equation:

$$v'_{ui} = \overline{u} + \frac{\sum_{r \in N} \mathrm{sim}(u, r) \times (v_{ri} - \overline{r})}{\sum_{v \in N} \mathrm{sim}(u, v)}, \tag{6}$$

where $\overline{u}$ is the average rating of items by user u, $\mathrm{sim}(u, r)$ is the similarity between user u and user r, v_{ri} is the rating of item i by user r, and $\overline{r}$ is the average rating of items by user r. Then, sort the items that user i did not score or purchase according to the predicted score, and obtain the Top-N items as the recommendation data set and recommend them to user i.

5.2. Collaborative Filtering Algorithm Based on NMF. The collaborative filtering algorithm based on NMF proposed in this paper can be divided into two processes: matrix factorization with dimensionality reduction and collaborative filtering.

(1) Matrix factorization and dimension reduction

Step 1: Using GPU-based NMF, the large-scale user preference matrix V is approximated by the product of two matrices W and H. The base matrix W stands for the item feature matrix, which contains the reduced set of r factors (r is the rank in NMF), and the projection matrix H stands for the user feature matrix, which stores the coefficient of the linear combination of the r factors.

Step 2: According to matrix W, the projection vector of the target user u_i's rating vector corresponding to the base matrix W can be calculated and denoted as h_i.

However, choosing a suitable number of latent factors will have an impact on the effect of NMF. In this paper, in order to improve the collaborative filtering-based recommendation, we need to select the optimal rank r for NMF. According to the cophenetic correlation coefficient [51], we repeat NMF several times per rank and calculate how similar the results are and, in other words, how stable the identified clusters are, given that the initial seed is random. We choose the highest r before the cophenetic coefficient drops.

(2) Collaborative filtering

Step 1: For the GPU-accelerated user similarity calculation, each user is assigned a thread in CUDA programming model, a kernel function is designed to calculate the Pearson correlation coefficient, and the similarity between h_i and each column of the projection matrix H is calculated in parallel to obtain the user similarity matrix *sim*.

Step 2: Top k users with the highest value of similarity form the nearest neighbor set N for user u_i.

Step 3: Use the neighbors of u_i in the nearest neighbor set N and the corresponding original scores in V to perform weighted calculation to generate the score prediction matrix p.

Step 4: Sort and get Top-N recommendation result using the prediction matrix p.

6. Performance Evaluations

6.1. Experiment Setups. For our experiments, we have used four n1-standard-4 instances of Google Compute Engine, and each instance is configured with 4 vCPU, 15 GB memory, and 100 GB SSD hard disk in asia-east1 district. Each instance is also configured with a NVIDIA K80 GPU with 2496 CUDA cores and 12 GB global memory. In the 4-node cluster, 64-bit Ubuntu 16.04 LTS is installed, and other software packages include Hadoop 2.7, Spark 2.3, JDK 1.8, and CUDA 9.0.

6.2. Data Set. MovieLens data set (https://grouplens.org/datasets/movielens/) provided by the GroupLens research group is used in the experiment. It contains the scores of 130,642 movies scored by 7,120 users. We randomly select data sets of different sizes for testing. Each user must rate at least 20 movies, the range of ratings is from 1 to 5, and the higher the rating is the more satisfied the user was. In the experiment, the movie ratings are converted into a scoring matrix. If a user does not rate a movie, the corresponding matrix element value is 0; thus the scoring matrix is a typical sparse matrix. In our experiment, the data set V we randomly selected needs to be further divided into a training set VT and a test set T through splitting the nonzero elements,

and 70% of the data set as the training set and the other 30% as the test set, making sure that matrix VT and matrix T have the same size as that of matrix V.

6.3. Baseline Algorithms

Serial NMF. Serial NMF algorithm is performed in a single thread using CPU only. According to equation (3) and (4), the method of alternately updating W and H is used to obtain the decomposition results by performing multiple iterations.

GPU-based NMF. GPU-based NMF algorithm is also performed in a single thread but with one GPU device support. As you see in Figure 3, alternately updating W and H is accelerated by GPU, implemented using the libraries of Cublas and Cusparse, together with two self-defined operations, dot multiplication and dot division.

Spark-based NMF without GPU support. For this algorithm, NMF is computed in a Spark cluster, and each node has no GPU device. Similar to Algorithm 1, in the two stages of *rddH.mapPartition* and *rddW.mapPartition*, there is no GPU support for the updating of H and W and only CPU for matrix operations in each iteration.

6.4. Evaluation Metrics for Recommendation. In order to accurately measure the performance of algorithms, in addition to the running time, the accuracies of prediction scores and recommendation results are also considered. In this paper, we use root mean square error (RMSE) and mean absolute error (MAE) to measure the accuracy of prediction scores. For the measurement of the accuracy of recommended results, the accuracy rate (Precision) and the recall rate (Recall) are generally used for measuring, together with F-measure for comprehensive consideration of contradictions between the two indicators.

RMSE measures the accuracy of predictions based on the root mean square error between the predicted score and the actual score. The smaller the value of RMSE, the more accurate the prediction result and the higher the quality of the recommended algorithm. The prediction score set $p = \{p_1, p_2, \ldots, p_n\}$ is obtained through training, and the actual user preference score set $T = \{t_1, t_2, \ldots, t_n\}$ is in the test set. Therefore, RMSE is defined as

$$\text{RMSE} = \sqrt{\sum_{i=1}^{n} \frac{(p_i - t_i)^2}{n}}. \tag{7}$$

MAE measures the accuracy of predictions based on the average deviation between the predicted score and the actual score, which is defined as

$$\text{MAE} = \frac{\sum_{i=1}^{n} |p_i - t_i|}{n}. \tag{8}$$

In our experiments, we use three evaluation metrics to evaluate the performance: *Precision*, *Recall*, and *F-measure*. Among the items that have never been purchased or rated, N items with the highest predicted ratings are selected to form the Top-N recommendation list. We define R_u as the set of items recommended for user u and define T_u as the set of items actually liked by user u in the test set. Accuracy means the proportion of related items in the recommended items. Simply speaking, it is recommendation hit rate (the hit means the recommended item has a score in the test set and the score exceeds a certain threshold). We define U as the set of all users, and the recommended accuracy is defined as Precision:

$$\text{Precision} = \frac{\sum_{u \in U} |R_u \cap T_u|}{\sum_{u \in U} |R_u|}. \tag{9}$$

The ratio of the correct recommended items to all items in the recommendation results is defined as Recall:

$$\text{Recall} = \frac{\sum_{u \in U} |R_u \cap T_u|}{\sum_{u \in U} |T_u|}. \tag{10}$$

F-measure is weighted harmonic average of Precision and Recall, which is defined as follows:

$$F - \text{measure} = \frac{\text{Precision} \times \text{Recall} \times 2}{\text{Precision} + \text{Recall}}. \tag{11}$$

In this paper, the training data VT is factorized by NMF, and then Top-N recommendation results are generated according to the algorithms in Subsection 5.2. The test data T is only used for calculating various recommendation evaluation metrics, such as RMSE, MAE, *Precision*, and *Recall*, without projecting the test data in the latent space created by the training data.

6.5. Result Analysis of NMF. In the experiments, we conducted performance evaluations using four algorithms: (i) Serial NMF, (ii) GPU-based NMF, (iii) Spark-based NMF without GPU support, and (iv) Spark-based NMF with GPU support which is proposed in this paper and developed on Spark-GPU fusion platform. We designed three performance comparisons to validate the new proposed algorithm. We select some typical matrix dimensions, and the number of iterations is 100.

6.5.1. Performance of GPU Speedup. We performed GPU-based NMF in a single node, and we varied the matrix dimensions as seen in Figure 4. We measured the computation time, and then we also performed the serial NMF in the same node so as to calculate the GPU speedup to validate the effectiveness of GPU acceleration. The speedup is defined as the ratio of the computation time of the single-node serial method to the computation time of the single-node GPU method; that is, Speedup = $(T_{\text{serial}}/T_{gpu\_\text{parallel}})$. The speedup varies with matrix dimensions, and we have obtained maximum speedup of 45x for GPU when compared with CPU.

6.5.2. Performance of NMF on Spark. In this evaluation, we started the Spark cluster, and the number of worker nodes is varied from 1, 2, and 3 to 4. We varied the matrix dimensions

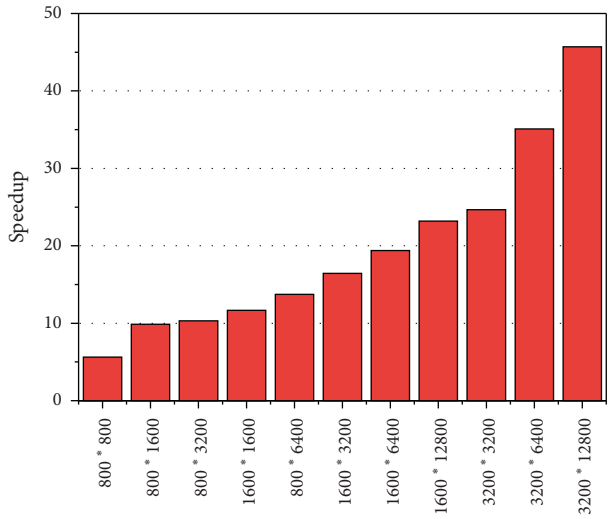

Figure 4: Performance of GPU speedup.

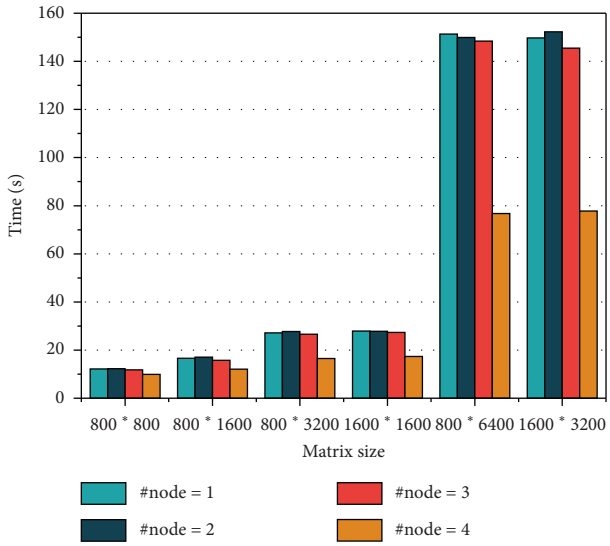

Figure 5: Performance of NMF on Spark.

from 800 ∗ 800, 800 ∗ 1600, 800 ∗ 3200, 1600 ∗ 1600, and 800 ∗ 6400 to 1600 ∗ 3200 and measured the computation time of NMF in Spark platform, and results are shown in Figure 5. When the number of nodes is 4, we set the number of Spark executors to 16, and, with the increase of the matrix dimensions, the advantages of 4 nodes are becoming more and more obvious. Compared with 3-node Spark platform, the computation time of 4 nodes saves about 50% of the time.

6.5.3. Performance of NMF on Spark with GPU Support. In the last evaluation, we started the Spark cluster, the number of nodes is 4, and we varied the matrix dimensions from 6400 ∗ 6400, 3200 ∗ 25600, and 6400 ∗ 12800 to 6400 ∗ 25600 and compared GPU support with non-GPU support. As can be seen from Figure 6, in the 4-node Spark platform, the computation time of NMF with GPU is smaller than that of NMF without GPU. When the size of matrix is 6400 ∗ 25600, NMF on Spark with GPU support saves about 10.8% of the time. NMF on GPU-accelerated Spark platform obviously shows execution efficiency.

Due to the mathematical fundamental of NMF and the blockwise-based parallel principle, there are frequent data distributions and data collections among all executors, and the communication cost is very high for the NMF on Spark. However, compared with data distributions and data collections, the execution of mapPartition function takes much less time due to the GPU acceleration. From the perspective of time analysis, communication and data exchange are the bottlenecks of NMF parallel algorithm. NMF on GPU-accelerated Spark platform still has great potential for improvement.

6.6. Result Analysis of Collaborative Filtering. In the experiments, we compared the performance of three algorithms: traditional user-based CF, traditional item-based CF, and the NMF-based CF proposed in this paper. The size of the matrices changes from 400 ∗ 800, 400 ∗ 1600, 800 ∗ 1600, and 800 ∗ 3200 to 1200 ∗ 3200 for testing. The number of iterations is 100, and we select 50 items for each user as the Top-50 recommendation list.

6.6.1. Comparison of Score Prediction Accuracy. In order to compare the prediction accuracy, we compared RMSE and MAE of the three algorithms, as shown in Figures 7(a) and 7(b), respectively. Under five different score matrix sizes, NMF-CF is significantly better than User-CF and Item-CF in terms of both RMSE and MAE. The results of RMSE and MAE for NMF-CF are the smallest, while the Item-CF algorithm obtains the largest prediction error and the worst prediction effect. When the size of the matrix is 400∗800, compared with the Item-CF algorithm, the result of RMSE for NMF-CF is reduced by 31.64%, and the MAE for NMF-CF is reduced by 28.5%.

6.6.2. Comparison of Recommendation Accuracy. The recommendation performances of Precision, Recall, and *F*-measure of the three algorithms are shown in Figures 8(a)–8(c), respectively. Under the five different score matrix sizes, as the size of the matrix increases, all the indicators for three algorithms have declined. NMF-CF is superior to User-CF and Item-CF algorithms in all three indicators, which explicitly shows that the quality of CF recommendations based on NMF is the best. However, with the increase of the matrix size $n \times m$, especially the increase of m, it means that the number of items increases, and we only recommend 50 items in collaborative filtering. When calculating the two indicators Precision and Recall, we compare with the 30% test set, and the hit rate of recommended items will be lower, and the advantage of NMF is getting smaller and smaller. When the matrix size is 1600 ∗ 3200, the results of NMF-CF and User-CF algorithm are almost the same. The recommended effect of the Item-CF algorithm has always been the worst.

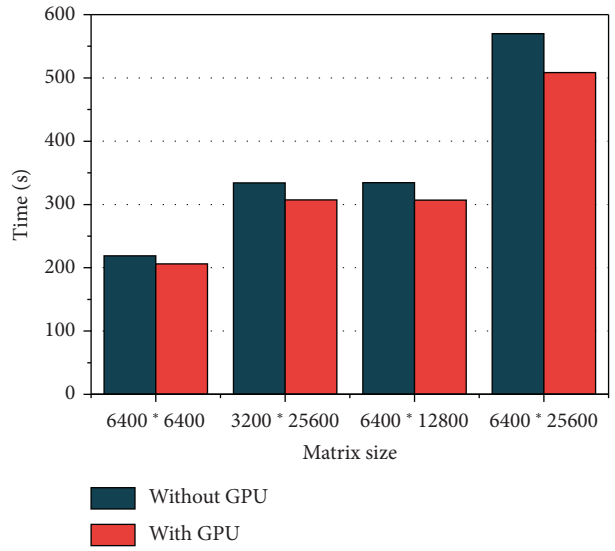

Figure 6: Performance of NMF on Spark with/without GPU support.

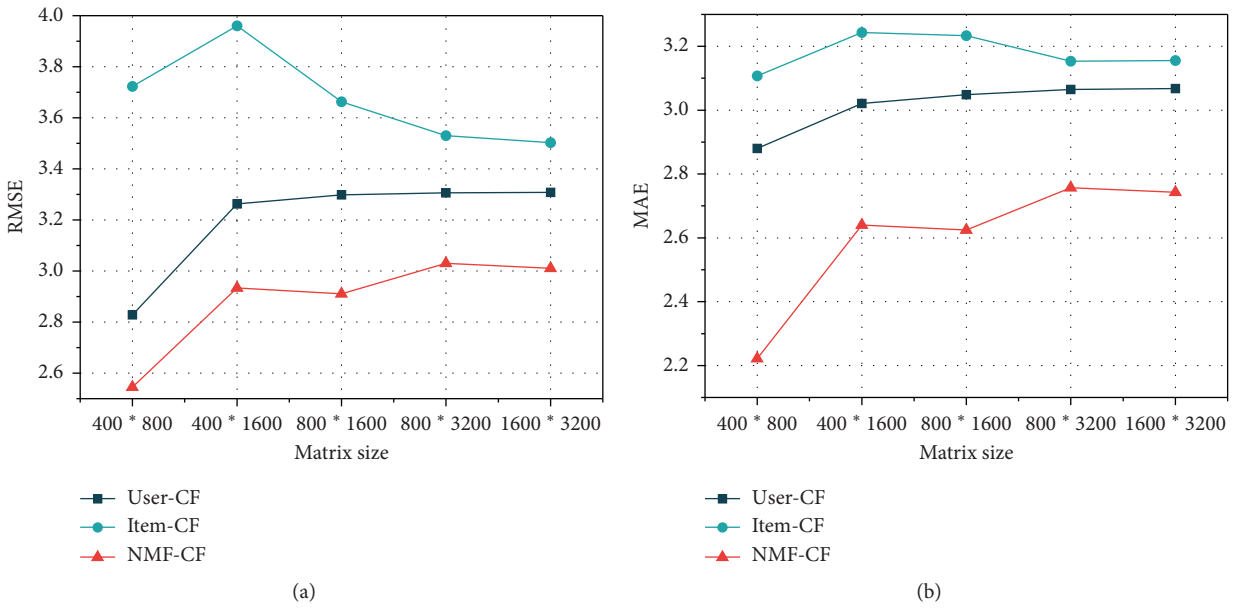

Figure 7: Score prediction accuracy results. (a) Comparison of RMSE and (b) comparison of MAE.

6.6.3. Comparison of Running Time for Recommendation. First, we only evaluated the running time of NMF-CF algorithm, and we considered two conditions in Spark platform: (i) CPU-based NMF-CF (only 1 node used) and (ii) CPU + GPU-based NMF-CF (4 nodes used). In the scenarios of five matrix sizes, the result of the running time is shown in Figure 9. It can be seen from the figure that when the GPU acceleration is adopted, the computation time for NMF is significantly reduced. As the matrix size becomes larger, the parallel efficiency is getting higher, and the acceleration effect is also getting better. When the matrix size is $1600 * 3200$, due to the utilization of GPU, CPU + GPU-based approach is reduced by 44.8% compared to CPU-based approach, which also proved the acceleration performance of GPU for NMF-CF.

Then, the running time comparison has been performed in the 4-node Spark platform with GPU support for the three algorithms. In all three algorithms, GPU is used to calculate Pearson correlation coefficient, and NMF-CF algorithm uses GPU to calculate NMF. The running time comparison results are shown in Figure 10. It can be seen from the figure

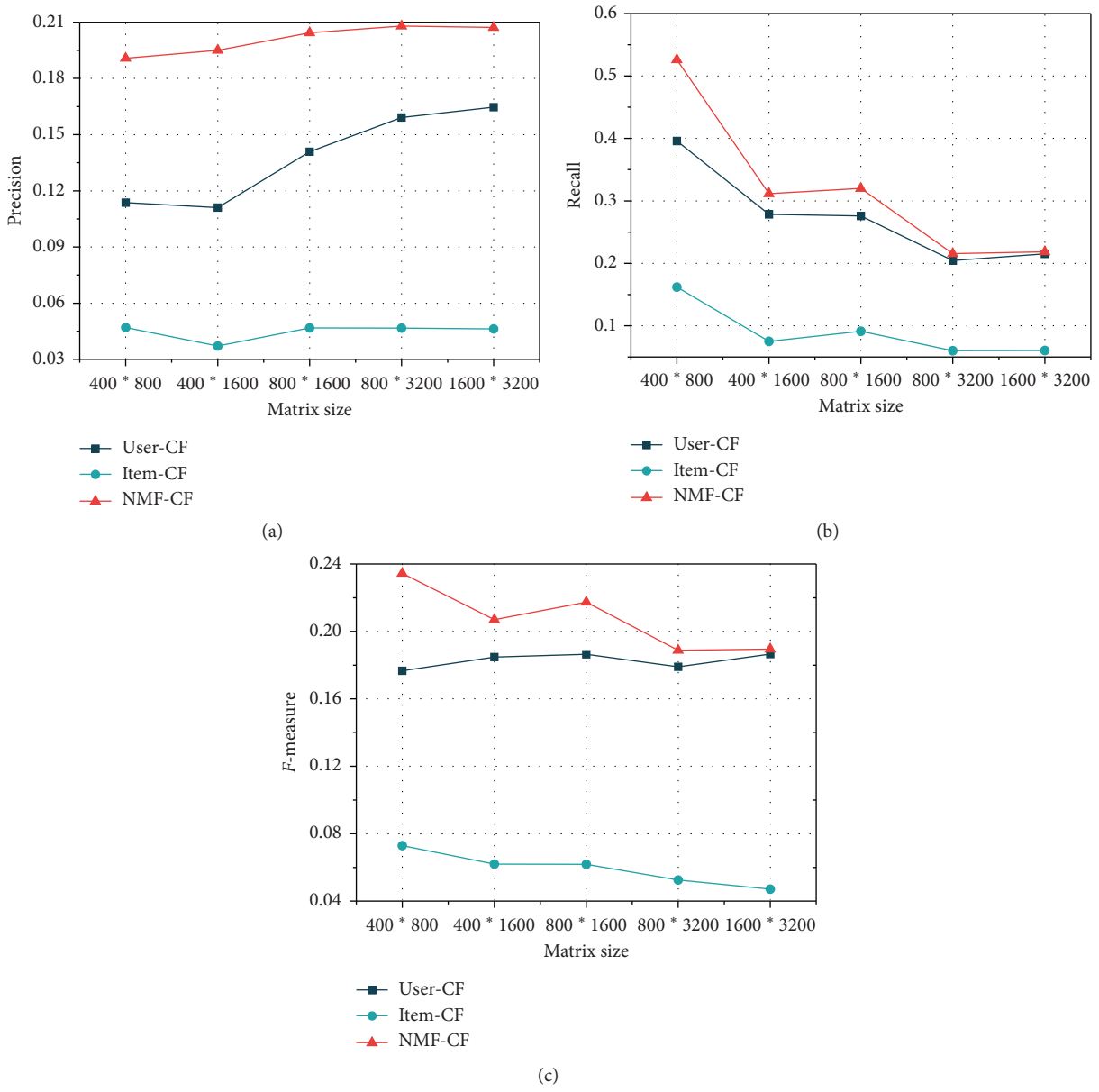

Figure 8: Recommendation accuracy results. (a) Comparison of Precision, (b) comparison of Recall, and (c) comparison of *F*-measure.

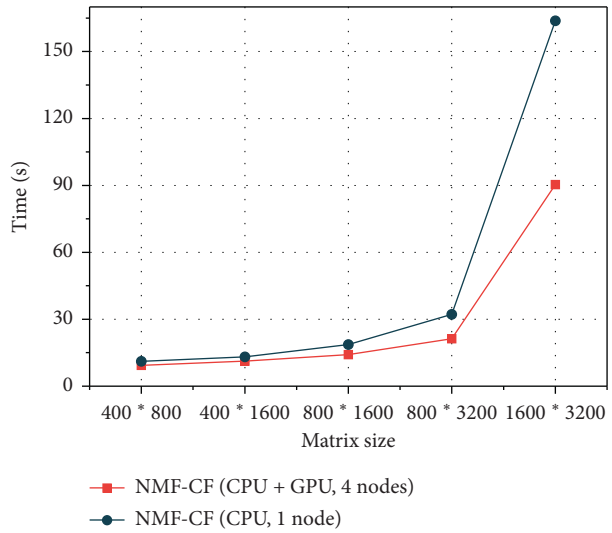

Figure 9: Running time comparison of NMF-CF under two conditions.

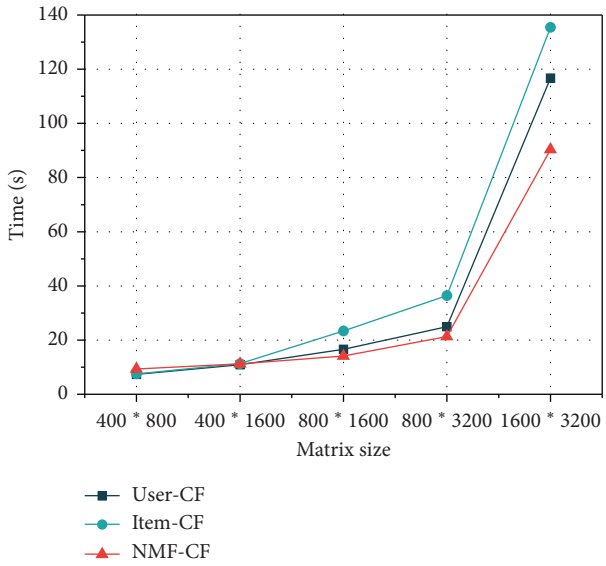

Figure 10: Running time comparison of three CF recommendation algorithms.

that, with the increase of the matrix size, the running time of each algorithm has increased. The running time of the NMF-CF recommendation algorithm significantly outperforms the User-CF and Item-CF algorithms. When the matrix size is $1600 * 3200$, the running time of the NMF-CF algorithm is reduced by 33.3% compared with the Item-CF algorithm and is reduced by 22.5% compared to the User-CF algorithm.

Overall, compared with the traditional CF, the NMF-CF recommendation algorithm contains the process of the decomposition of the scoring matrix V into W and H, which seems to be a time-consuming operation. In fact, when calculating the correlation coefficient later, it will save a lot of time for NMF-CF. Since the size of matrix W is $n \times r$, where the value of r reflects the number of features or topics, the value of r is usually very small (it generally takes a value of 2 to 10), and the size of matrix W is small, so through calculating the correlation coefficient to obtain k-nearest neighbors for each user takes much less time in NMF-CF algorithm than in the User-CF algorithm or Item-CF algorithm. In addition, except the increased accuracy, NMF-based CF recommendation algorithm uses GPUs to run in parallel and the elapsed computation time is still the shortest.

7. Conclusion

In the heterogeneous CPU/GPU cluster, nodes have large memory resources and GPU multicore resources, and the advantages of distributed storage between nodes and data sharing within nodes should be utilized. Heterogeneous parallel computing is an efficient and feasible parallel programming strategy. A GPU-accelerated NMF algorithm on Spark platform has been designed in this paper to solve the problem of low processing speed of NMF as the size of the matrix increases. Through the performance evaluations, experimental results have proved that the combination of Spark-based in-memory computing and GPU has higher execution efficiency. On the other hand, recommendation systems have been widely applied in many fields, but as the user number and item number increase, the computational speed also becomes slower and the accuracy of recommendation decreases. Although traditional collaborative filtering is extremely successful in the recommendation system, as the data increases, the recommendation algorithms have been confronted with various problems, such as scalability problems, cold start problems, and matrix sparseness problems. This paper implemented the NMF algorithm for collaborative filtering recommendation, which combines NMF with traditional collaborative filtering methods, decomposes the original score data into base matrix and projection matrix, and runs in parallel on Spark platform accelerated by GPU. Experiments on matrices with different size show that the parallel NMF collaborative filtering recommendation algorithm not only improves the prediction and recommendation accuracy but also greatly improves the calculation efficiency.

Data Availability

The data used to support the findings of this study are available from the corresponding author upon request.

Conflicts of Interest

The authors declare that they have no conflicts of interest.

Acknowledgments

This work was supported by the National Natural Science Foundation of China under Grants nos. 61602169 and 61872138, the National Key R&D Program of China under Grant no. 2018YFB1402800, and the Natural Science Foundation of Hunan Province under Grant no. 2018JJ2135, as well as the Scientific Research Fund of Hunan Provincial Education Department under Grant no. 18A186.

References

[1] D. D. Lee and H. S. Seung, "Learning the parts of objects by non-negative matrix factorization," *Nature*, vol. 401, no. 6755, pp. 788–791, 1999.

[2] D. D. Lee and H. S. Seung, "Algorithms for non-negative matrix factorization," in *Advances in Neural Information Processing Systems 13, Papers from Neural Information Processing Systems (NIPS) 2000*, T. K. Leen, T. G. Dietterich, and V. Tresp, Eds., pp. 556–562, MIT Press, Denver, CO, USA, 2000.

[3] A. Falini, G. Castellano, C. Tamborrino et al., "Saliency detection for hyperspectral images via sparse-non negative-matrix-factorization and novel distance measures," in *Proceedings of the 2020 IEEE Conference on Evolving and Adaptive Intelligent Systems, EAIS 2020*, pp. 1–8, IEEE, Bari, Italy, 2020.

[4] V. Leplat, N. Gillis, and A. M. S. Ang, "Blind audio source separation with minimum-volume beta-divergence NMF,"

IEEE Transactions on Signal Processing, vol. 68, pp. 3400–3410, 2020.

[5] X. Luo, M. Zhou, Y. Xia, and Q. Zhu, "An efficient non-negative matrix-factorization-based approach to collaborative filtering for recommender systems," *IEEE Transactions on Industrial Informatics*, vol. 10, no. 2, pp. 1273–1284, 2014.

[6] S. Rendle and L. Schmidt-Thieme, "Online-updating regularized kernel matrix factorization models for large-scale recommender systems," in *Proceedings of the 2008 ACM Conference on Recommender Systems, RecSys 2008*, P. Pu, D. G. Bridge, B. Mobasher, and F. Ricci, Eds., pp. 251–258, ACM, Lausanne, Switzerland, 2008.

[7] Y. Chen, M. Rege, M. Dong, and J. Hua, "Non-negative matrix factorization for semi-supervised data clustering," *Knowledge and Information Systems*, vol. 17, no. 3, pp. 355–379, 2008.

[8] J. Choo, C. Lee, C. K. Reddy, and H. Park, "UTOPIAN: user-driven topic modeling based on interactive nonnegative matrix factorization," *IEEE Transactions on Visualization and Computer Graphics*, vol. 19, no. 12, pp. 1992–2001, 2013.

[9] V. Kysenko, K. Rupp, O. Marchenko, S. Selberherr, and A. Anisimov, "GPU-accelerated non-negative matrix factorization for text mining," in *Natural Language Processing and Information Systems—17th International Conference on Applications of Natural Language to Information Systems, NLDB 2012, Groningen, The Netherlands, June 26–28, 2012. Proceedings, Lecture Notes in Computer Science*, G. Bouma, A. Ittoo, E. Métais, and H. Wortmann, Eds., vol. 7337, pp. 158–163, Springer, Berlin, Germany, 2012.

[10] R. Liao, Y. Zhang, J. Guan, and S. Zhou, "CloudNMF: a mapreduce implementation of nonnegative matrix factorization for large-scale biological datasets," *Genomics, Proteomics & Bioinformatics*, vol. 12, no. 1, pp. 48–51, 2014.

[11] S. Mittal and J. S. Vetter, "A survey of CPU-GPU heterogeneous computing techniques," *ACM Comput. Surv.*vol. 47, no. 4, pp. 1–69, 2015.

[12] M. Zaharia, M. Chowdhury, M. J. Franklin, S. Shenker, and I. Stoica, "Spark: cluster computing with working sets," in *2nd USENIX Workshop on Hot Topics in Cloud Computing, HotCloud'10*, E. M. Nahum and D. Xu, Eds., USENIX Association, Boston, MA, USA, 2010.

[13] M. Zaharia, R. S. Xin, P. Wendell et al., "Apache spark," *Communications of the ACM*, vol. 59, no. 11, pp. 56–65, 2016.

[14] M. Zaharia, M. Chowdhury, T. Das et al., "Resilient distributed datasets: a fault-tolerant abstraction for in-memory cluster computing," in *Proceedings of the 9th USENIX Symposium on Networked Systems Design and Implementation, NSDI 2012*, pp. 15–28, USENIX Association, San Jose, CA, USA, 2012.

[15] J. Dean and S. Ghemawat, "MapReduce," *Communications of the ACM*, vol. 51, no. 1, pp. 107–113, 2008.

[16] B. Tang, M. Tang, G. Fedak, and H. He, "Availability/network-aware mapreduce over the internet," *Information Sciences*, vol. 379, pp. 94–111, 2017.

[17] R. Kannan, G. Ballard, and H. Park, "A high-performance parallel algorithm for nonnegative matrix factorization," in *Proceedings of the 21st ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming, PPoPP 2016*pp. 1–11, Barcelona, Spain, 2016.

[18] E. Mejía-Roa, D. Tabas-Madrid, J. Setoain, C. García, F. Tirado, and A. D. Pascual-Montano, "NMF-mGPU: non-negative matrix factorization on multi-GPU systems," *BMC Bioinformatics*, vol. 16, pp. 1–43, 2015.

[19] C. Liu, H. Yang, J. Fan, L. He, and Y. Wang, "Distributed nonnegative matrix factorization for web-scale dyadic data analysis on mapreduce," in *Proceedings of the 19th International Conference on World Wide Web, WWW 2010*, M. Rappa, P. Jones, J. Freire, and S. Chakrabarti, Eds., pp. 681–690, ACM, Raleigh, NC, USA, 2010.

[20] H. Jiang, Y. Chen, Z. Qiao, K.-C. Li, W. Ro, and J.-L. Gaudiot, "Accelerating mapreduce framework on multi-GPU systems," *Cluster Computing*, vol. 17, no. 2, pp. 293–301, 2014.

[21] T. Gao, Y. Guo, B. Zhang et al., "Memory-efficient and skew-tolerant mapreduce over MPI for supercomputing systems," *IEEE Transactions on Parallel and Distributed Systems*, vol. 31, no. 12, pp. 2734–2748, 2020.

[22] M. Sergent, M. Dagrada, P. Carribault, J. Jaeger, M. Pérache, and G. Papauré, "Efficient communication/computation overlap with MPI+OpenMP runtimes collaboration," in *Euro-Par 2018: Parallel Processing - 24th International Conference on Parallel and Distributed Computing, Turin, Italy, August 27-31, 2018, Proceedings, Lecture Notes in Computer Science*, M. Aldinucci, L. Padovani, and M. Torquati, Eds., vol. 11014, pp. 560–572, Springer, Berlin, Germany, 2018.

[23] J. A. Stuart and J. D. Owens, "Multi-GPU mapreduce on GPU clusters," in *25th IEEE International Symposium on Parallel and Distributed Processing, IPDPS 2011*, pp. 1068–1079, IEEE, Anchorage, AK, USA, 2011.

[24] N. Guan, D. Tao, Z. Luo, and B. Yuan, "Online nonnegative matrix factorization with robust stochastic approximation," *IEEE Trans. Neural Networks Learn. Syst.*vol. 23, no. 7, pp. 1087–1099, 2012.

[25] D. Tu, L. Chen, M. Lv, H. Shi, and G. Chen, "Hierarchical online NMF for detecting and tracking topic hierarchies in a text stream," *Pattern Recognition*, vol. 76, pp. 203–214, 2018.

[26] N. Halko, P. G. Martinsson, and J. A. Tropp, "Finding structure with randomness: Probabilistic algorithms for constructing approximate matrix decompositions," *SIAM Review*, vol. 53, no. 2, pp. 217–288, 2011.

[27] M. Tepper and G. Sapiro, "Compressed nonnegative matrix factorization is fast and accurate," *IEEE Transactions on Signal Processing*, vol. 64, no. 9, pp. 2269–2283, 2016.

[28] A. Janecek, S. S. Grotthoff, and W. N. Gansterer, "libNMF—a library for nonnegative matrix factorization," *Computing and Informatics*, vol. 30, no. 2, pp. 205–224, 2011.

[29] N. Lopes and B. Ribeiro, "Non-negative matrix factorization implementation using graphic processing units," in *Intelligent Data Engineering and Automated Learning - IDEAL 2010, 11th International Conference, Paisley, UK, September 1-3, 2010. Proceedings, Lecture Notes in Computer Science*, C. Fyfe, P. Tiňo, D. Charles et al., Eds., vol. 6283, pp. 275–283, Springer, Berlin, Germany, 2010.

[30] E. Battenberg and D. Wessel, "Accelerating non-negative matrix factorization for audio source separation on multi-core and many-core architectures," in *Proceedings of the 10th International Society for Music Information Retrieval Conference, ISMIR 2009, Kobe International Conference Center*, K. Hirata, G. Tzanetakis, and K. Yoshii, Eds., pp. 501–506, International Society for Music Information Retrieval, Kobe, Japan, 2009.

[31] B. Tang, L. Bobelin, and H. He, "Parallel algorithm of non-negative matrix factorization based on hybrid MPI and OpenMP programming model," *Computer Science*, vol. 44, no. 3, pp. 51–54, 2017.

[32] Z. Sun, T. Li, and N. Rishe, "Large-scale matrix factorization using mapreduce," in *ICDMW 2010, The 10th IEEE International Conference on Data Mining Workshops*, W. Fan, W. Hsu, G. I. Webb et al., Eds., pp. 1242–1248, IEEE Computer Society, Sydney, Australia, 2010.

[33] B. Tang, L. Kang, Y. Xia, and L. Zhang, "GPU-accelerated large-scale non-negative matrix factorization using spark," in *Collaborative Computing: Networking, Applications and Worksharing—14th EAI International Conference, CollaborateCom 2018, Shanghai, China, December 1–3, 2018, Proceedings, Lecture Notes of the Institute for Computer Sciences, Social Informatics and Telecommunications Engineering*, H. Gao, X. Wang, Y. Yin, and M. Iqbal, Eds., vol. 268, pp. 189–201, Springer, Berlin, Germany, 2018.

[34] B. Kwon and H. Cho, "Scalable co-clustering algorithms," in *Algorithms and Architectures for Parallel Processing, 10th International Conference, ICA3PP 2010, Busan, Korea, May 21–23, 2010. Proceedings. Part I, Lecture Notes in Computer Science*, C. Hsu, L. T. Yang, J. H. Park, and S. Yeo, Eds., vol. 6081, pp. 32–43, Springer, Berlin, Germany, 2010.

[35] Z. Liu, Y. Zhang, E. Y. Chang, and M. Sun, "PLDA+: parallel latent dirichlet allocation with data placement and pipeline processing," *ACM Trans. Intell. Syst. Technol.*vol. 2, no. 3, pp. 1–26, 2011.

[36] E. Karydi and K. G. Margaritis, "Multithreaded implementation of the slope one algorithm for collaborative filtering," in *Artificial Intelligence Applications and Innovations - 8th IFIP WG 12.5 International Conference, AIAI 2012, Halkidiki, Greece, September 27–30, 2012, Proceedings, Part I, IFIP Advances in Information and Communication Technology*, L. S. Iliadis, I. Maglogiannis, and H. Papadopoulos, Eds., vol. 381, pp. 117–125, Springer, Berlin, Germany, 2012.

[37] H. Yu, C. Hsieh, S. Si, and I. S. Dhillon, "Scalable coordinate descent approaches to parallel matrix factorization for recommender systems," in *12th IEEE International Conference on Data Mining, ICDM 2012*, M. J. Zaki, A. Siebes, J. X. Yu, B. Goethals, G. I. Webb, and X. Wu, Eds., pp. 765–774, IEEE Computer Society, Brussels, Belgium, 2012.

[38] K. Kato and T. Hosino, "Solving k-nearest neighbor problem on multiple graphics processors," in *Proceedings of the 10th IEEE/ACM International Conference on Cluster, Cloud and Grid Computing, CCGrid 2010, 17–20 May 2010*, pp. 769–773, IEEE Computer Society, Melbourne, Victoria, Australia, 2010.

[39] Z. Wang, Y. Liu, and S. Chiu, "An efficient parallel collaborative filtering algorithm on multi-GPU platform," *The Journal of Supercomputing*, vol. 72, no. 6, pp. 2080–2094, 2016.

[40] J. Jiang, J. Lu, G. Zhang, and G. Long, "Scaling-up item-based collaborative filtering recommendation algorithm based on hadoop," in *World Congress on Services, SERVICES 2011*, pp. 490–497, IEEE Computer Society, Washington, DC, USA, 2011.

[41] S. Schelter, C. Boden, and V. Markl, "Scalable similarity-based neighborhood methods with mapreduce," in *Sixth ACM Conference on Recommender Systems, RecSys '12*, P. Cunningham, N. J. Hurley, I. Guy, and S. S. Anand, Eds., pp. 163–170, ACM, Dublin, Ireland, 2012.

[42] A. Narang, A. Srivastava, and N. P. K. Katta, "Distributed scalable collaborative filtering algorithm," in *Euro-Par 2011 Parallel Processing—17th International Conference, Euro-Par 2011, Bordeaux, France, August 29—September 2, 2011, Proceedings, Part I, Lecture Notes in Computer Science*, E. Jeannot, R. Namyst, and J. Roman, Eds., vol. 6852, pp. 353–365, Springer, Berlin, Germany, 2011.

[43] E. Karydi and K. G. Margaritis, "Parallel implementation of the slope one algorithm for collaborative filtering," in *Proceedings of the 16th Panhellenic Conference on Informatics, PCI 2012*, pp. 174–179, IEEE Computer Society, Piraeus, Greece, 2012.

[44] Y. Koren, R. Bell, and C. Volinsky, "Matrix factorization techniques for recommender systems," *Computer*, vol. 42, no. 8, pp. 30–37, 2009.

[45] S. Gong, H. Ye, and Y. Dai, "Combining singular value decomposition and item-based recommender in collaborative filtering," in *Proceedings of the Second International Workshop on Knowledge Discovery and Data Mining, WKDD 2009*, pp. 769–772, IEEE Computer Society, Moscow, Russia, 2009.

[46] H. Polat and W. Du, "SVD-based collaborative filtering with privacy," in *Proceedings of the 2005 ACM Symposium on Applied Computing (SAC)*, H. Haddad, L. M. Liebrock, A. Omicini, and R. L. Wainwright, Eds., pp. 791–795, ACM, Santa Fe, NM, USA, 2005.

[47] K. Goldberg, T. Roeder, D. Gupta, and C. Perkins, "Eigentaste: a constant time collaborative filtering algorithm," *Information Retrieval*, vol. 4, no. 2, pp. 133–151, 2001.

[48] R. Salakhutdinov and A. Mnih, "Probabilistic matrix factorization," in *Advances in Neural Information Processing Systems 20, Proceedings of the Twenty-First Annual Conference on Neural Information Processing Systems*, J. C. Platt, D. Koller, Y. Singer, and S. T. Roweis, Eds., pp. 1257–1264, Curran Associates, Inc., Vancouver, British Columbia, Canada, 2007.

[49] N. Thai-Nghe, L. Drumond, T. Horváth, A. Nanopoulos, and L. Schmidt-Thieme, "Matrix and tensor factorization for predicting student performance," in *CSEDU 2011—Proceedings of the 3rd International Conference on Computer Supported Education*, A. Verbraeck, M. Helfert, J. Cordeiro, and B. Shishkov, Eds., vol. 1, pp. 69–78, SciTePress, Noordwijkerhout, Netherlands, 2011.

[50] C. Boutsidis and E. Gallopoulos, "SVD based initialization: a head start for nonnegative matrix factorization," *Pattern Recognition*, vol. 41, no. 4, pp. 1350–1362, 2008.

[51] J.-P. Brunet, P. Tamayo, T. R. Golub, and J. P. Mesirov, "Metagenes and molecular pattern discovery using matrix factorization," *Proceedings of the National Academy of Sciences*, vol. 101, no. 12, pp. 4164–4169, 2004.