Distributed Stochastic ADMM for Matrix Factorization

Zhi-Qin Yu
Shanghai Key Laboratory of Scalable Computing and Systems
Department of Computer Science and Engineering
Shanghai Jiao Tong University, China
yzqfqyd@sjtu.edu.cn

Xing-Jian Shi
Department of Computer Science and Engineering
Hong Kong University of Science and Technology, China
xshiab@connect.ust.hk

Ling Yan
Shanghai Key Laboratory of Scalable Computing and Systems
Department of Computer Science and Engineering
Shanghai Jiao Tong University, China
yling0718@sjtu.edu.cn

Wu-Jun Li
National Key Laboratory for Novel Software Technology
Department of Computer Science and Technology
Nanjing University, China
liwujun@nju.edu.cn

ABSTRACT

Matrix factorization (MF) has become the most popular technique for recommender systems due to its promising performance. Recently, distributed (parallel) MF models have received much attention from researchers of big data community. In this paper, we propose a novel model, called distributed stochastic alternating direction methods of multipliers (DS-ADMM), for large-scale MF problems. DS-ADMM is a distributed stochastic variant of ADMM. In particular, we first devise a new data split strategy to make the distributed MF problem fit for the ADMM framework. Then, a stochastic ADMM scheme is designed for learning. Finally, we implement DS-ADMM based on message passing interface (MPI), which can run on clusters with multiple machines (nodes). Experiments on several data sets from recommendation applications show that our DS-ADMM model can outperform other state-of-the-art distributed MF models in terms of both efficiency and accuracy.

Keywords

Matrix Factorization, Recommender Systems, ADMM, Distributed Computing, Stochastic Learning

1. INTRODUCTION

Recommender systems try to recommend products (items) to customers (users) by utilizing the customers' historic preferences. Matrix factorization (MF) [8] and its extensions [9, 22, 16, 14, 10, 18] have become the most popular models for recommender systems due to their promising performance [8]. In this big data era, more and more large-scale data sets have emerged in many real-world recommender systems. Hence, parallel or distributed[1] MF models with the potential of high scalability have recently captured much attention from researchers.

The basic idea of MF is to use the multiplication of two latent matrices, the user matrix and the item matrix, to approximate the original rating matrix. Least square method is usually used to find a solution. In recent years, several parallel models have been proposed for MF. These existing models can be roughly divided into two main categories: alternating least square (ALS) [23] based models and stochastic gradient descent (SGD) based models.

ALS [23] adopts the alternating learning strategy to update one matrix with the other one fixed. With one of the matrices fixed, the optimization problem of MF can be reduced to a least square problem on the other matrix, which can be further decomposed into several independent least square problems on the latent feature vector of each user or item. Hence, it is easy to design parallel strategies for ALS, which has been implemented in [23]. However, the time complexity for each iteration in ALS is cubic in k, where k is the number of latent features for each user or item. The cyclic coordinate descent (CCD) method [13] adopts coordinate descent strategy to improve the ALS method by decreasing the time complexity for each iteration to be linear in k. The CCD++ [21] further improves the efficiency of CCD by using similar coordinate descent strategy but different updating sequence of the variables. Because both CCD and CCD++ are based on ALS, they can also be easily parallelized [21].

CIKM'14, November 3–7, 2014, Shanghai, China.

http://dx.doi.org/10.1145/2661829.2661996 .

[1]In existing literatures, *distributed models* refer to those implemented on clusters with multiple machines (nodes), while *parallel models* refer to those implemented either on multi-core systems with a single node or on clusters. We will also follow this tradition in this paper. The specific meaning of *parallel* can be determined from the context in the paper.

Due to its efficiency and ease of implementation, SGD has become one of the most popular optimization strategies for MF in recommender systems [8]. The basic idea of SGD is to randomly select one rating each time from the rating matrix and then use the gradient based on this selected rating to update the latent features. It is easy to see that SGD is essentially a sequential method, which makes it difficult to be parallelized.

The main reason that SGD can not be directly parallelized is that two randomly selected ratings may share the same latent features corresponding to the same user or item. Hence, there exist conflicts between two processes or nodes which simultaneously update the same latent features. Recently, several strategies have been proposed to parallelize SGD for MF. The Hogwild! [11] model just ignores the conflicts by assuming that the probability of conflict is small when two ratings are randomly selected from a sparse rating matrix. However, the conflicts do exist in the learning procedure, which makes Hogwild! not effective enough [21, 24]. Moreover, Hogwild! requires all the processes share the whole training set which is hard to be satisfied in distributed systems. Hence, Hogwild! cannot be directly used in distributed systems.

Distributed SGD (DSGD) [4] utilizes the property that there exist several sub-blocks without overlapping rows and columns in the rating matrix. These sub-blocks are mutually independent of each other, thus can be processed in parallel by different processes or nodes at the same time. Experiments in [21, 24] have shown that DSGD can outperform Hogwild! in terms of both efficiency and accuracy. However, after a set of independent sub-blocks have been processed, the updated variables from all processes or nodes should be *synchronized* before processing the other sets of independent sub-blocks. It is these frequent synchronization operations that make DSGD inefficient because the slowest node will become the bottleneck of the whole system. Things go even worse if data skew exists, which is not rare in real applications. Very recently, fast parallel SGD (FPSGD) [24] tries to solve the issues in DSGD by changing the scheduler into an asynchronous one, which has achieved better performance than DSGD. However, FPSGD can only be used in shared-memory systems with a single node. Hence, FPSGD is still not scalable to handle large-scale problems.

In this paper, a novel model, called distributed stochastic alternating direction methods of multipliers (DS-ADMM), is proposed for large-scale MF problems. DS-ADMM is a distributed stochastic variant of ADMM [3]. The main contributions of DS-ADMM are briefly outlined as follows:

- In DS-ADMM, a new data split (partition) strategy called *LocalMFSplit* is proposed to assign subsets of the whole set of ratings to different nodes in a cluster and consequently divide the large-scale problem into several sub-problems. Our split strategy can make the distributed MF problem fit for the ADMM framework. Furthermore, compared with existing split strategies in DSGD and CCD++, our split strategy can reduce synchronization and scheduling cost to improve efficiency.

- A stochastic ADMM method is designed to perform efficient learning for parameters.

- DS-ADMM is implemented with message passing interface (MPI), which can run on clusters with multiple machines (nodes). Hence, DS-ADMM is scalable to handle large-scale data sets.

- Experiments on several data sets from recommendation applications show that not only can DS-ADMM outperform other SGD-based models, but it can also outperform ALS-based models like CCD++ in terms of both efficiency and accuracy.

2. BACKGROUND

In this section, we introduce the background of this paper, including notations, MF formulation, ALS-based models, SGD-based models and ADMM.

2.1 Notations

We use boldface uppercase letters like $\mathbf{M}$ to denote matrices and boldface lowercase letters like $\mathbf{m}$ to denote vectors. $\mathbf{M}_{i*}$ and $\mathbf{M}_{*j}$ denote the ith row and the jth column of $\mathbf{M}$, respectively. M_{ij} denotes the element at the ith row and jth column in $\mathbf{M}$. $\mathbf{M}^T$ denotes the transpose of $\mathbf{M}$, and $\mathbf{M}^{-1}$ denotes the inverse of $\mathbf{M}$. $\mathrm{tr}(\cdot)$ denotes the trace of a matrix. $\mathbf{I}_k$ is an identity matrix of size $k \times k$. Assume there are m users and n items in the data set. We use $\mathbf{R} \in \mathbb{R}^{m \times n}$ to denote the rating matrix. Please note that there exist many missing entries in $\mathbf{R}$. All the missing entries are filled with 0. We use $\Omega \subset \{1, 2, \cdots, m\} \times \{1, 2, \cdots, n\}$ to denote the set of indices for the observed ratings. Ω_i denotes the column indices of the observed ratings in the ith row of $\mathbf{R}$, and $\tilde{\Omega}_j$ denotes the row indices of the observed ratings in the jth column of $\mathbf{R}$. $\mathbf{U} \in \mathbb{R}^{k \times m}$ denotes the users' latent factors (matrix) with each column $\mathbf{U}_{*i}$ representing the latent feature vector for user i, where k is the number of latent factors for each user or item. $\mathbf{V} \in \mathbb{R}^{k \times n}$ denotes the items' latent factors (matrix) with each column $\mathbf{V}_{*j}$ representing the latent feature vector for item j. P denotes the total number of nodes in the cluster, and we use the letter p on the superscript like $\mathbf{M}^p$ to denote the computer node id. $\|\cdot\|_F$ denotes the Frobenius norm of a matrix or a vector.

2.2 Matrix Factorization

Matrix factorization (MF) can be formulated as the following optimization problem:

$$\min_{\mathbf{U},\mathbf{V}} \frac{1}{2} \sum_{(i,j)\in\Omega} \left[(R_{i,j} - \mathbf{U}_{*i}^T \mathbf{V}_{*j})^2 + \lambda_1 \mathbf{U}_{*i}^T \mathbf{U}_{*i} + \lambda_2 \mathbf{V}_{*j}^T \mathbf{V}_{*j} \right], \tag{1}$$

where λ_1 and λ_2 are hyper-parameters for regularization.

There are two categories of parallel models to solve the above MF problem, i.e., the ALS-based models and SGD-based models, which will be briefly reviewed in the following subsections.

2.3 ALS-based Parallel MF Models

By adopting the alternating learning strategy, ALS [23] alternatively switches between updating $\mathbf{U}$ and updating $\mathbf{V}$ with the other latent matrix fixed. With $\mathbf{U}$ fixed, the MF problem can be decomposed into n independent least square problems, each of which corresponds to a column of the matrix $\mathbf{V}$. Similar m independent least square problems can be got by fixing $\mathbf{V}$. Furthermore, each of these independent

problems has a closed-form solution in ALS:

$$\mathbf{U}_{*i} \leftarrow (\lambda_1 m_i \mathbf{I}_k + \mathbf{V}_{\Omega_i}\mathbf{V}_{\Omega_i}^T)^{-1}\mathbf{V}\mathbf{R}_{i*}^T, \quad (2)$$
$$\mathbf{V}_{*j} \leftarrow (\lambda_2 n_j \mathbf{I}_k + \mathbf{U}_{\tilde{\Omega}_j}\mathbf{U}_{\tilde{\Omega}_j}^T)^{-1}\mathbf{U}\mathbf{R}_{*j},$$

where $\mathbf{V}_{\Omega_i}$ denotes a sub-matrix formed by the columns in $\mathbf{V}$ indexed by Ω_i, $\mathbf{U}_{\tilde{\Omega}_j}$ is similarly defined, $m_i = |\Omega_i|$ and $n_j = |\tilde{\Omega}_j|$. Please note that all the missing entries in $\mathbf{R}$ have been filled by zeros. The columns in both $\mathbf{U}$ and $\mathbf{V}$ can be independently updated by following (2). Hence, it is easy to design the parallel strategy for ALS, which has been implemented in [23].

Instead of optimizing the whole vector $\mathbf{U}_{*i}$ or $\mathbf{V}_{*j}$ at one time, CCD [13] adopts the coordinate descent method to optimize each element of $\mathbf{U}_{*i}$ or $\mathbf{V}_{*j}$ separately, which can avoid the matrix inverse operation in (2). CCD++ [21] further improves CCD's performance by changing the updating sequence in CCD. It rewrites $\mathbf{U}^T\mathbf{V} = \sum_{d=1}^{k}\mathbf{U}_{d*}^T\mathbf{V}_{d*}$, and updates one element in $\mathbf{U}_{d*}$ or $\mathbf{V}_{d*}$ each time by using similar coordinate descent method in CCD. Changing the updating sequence may improve the convergence rate, which has been verified by the experimental results in CCD++ [21].

2.4 SGD-based Parallel MF Models

The idea of SGD is to randomly select one rating index (i, j) from Ω each time, and then update the corresponding variables $\mathbf{U}_{*i}$ an $\mathbf{V}_{*j}$ as follows:

$$\mathbf{U}_{*i} \leftarrow \mathbf{U}_{*i} + \eta(\epsilon_{ij}\mathbf{V}_{*j} - \lambda_1\mathbf{U}_{*i}), \quad (3)$$
$$\mathbf{V}_{*j} \leftarrow \mathbf{V}_{*j} + \eta(\epsilon_{ij}\mathbf{U}_{*i} - \lambda_2\mathbf{V}_{*j}),$$

where $\epsilon_{ij} = R_{ij} - \mathbf{U}_{*i}^T\mathbf{V}_{*j}$, and η is the learning rate.

Due to the demand of many large-scale problems, several parallel SGD models have been proposed. Some of them, such as those described in [25] and [20], are not for MF problems. Here, we just focus on those parallel SGD models for MF, including Hogwild! [11], DSGD [4] and FPSGD [24].

From (3), it is easy to find that conflicts exist between two processes or nodes when their randomly selected ratings share either the same user index or the same item index. Hogwild! [11] allows overwriting each other's work when conflicts happen. It also shows that if the optimization problem is sparse enough, the Hogwild! will get a nearly optimal rate of convergence.

DSGD [4] divides the whole rating matrix into P strata and each stratum contains P mutually independent sub-blocks without sharing any column or row indices. Consequently, sub-blocks in the same stratum can be processed in parallel since they don't share any $\mathbf{U}_{*i}$ or $\mathbf{V}_{*j}$. One iteration of DSGD is divided into P steps, in each of which DSGD picks a stratum containing P independent sub-blocks and then processes these sub-blocks in parallel in a cluster of P nodes with each node responsible for one sub-block. After all the P sub-blocks in each step are processed, the whole $\mathbf{U}$ and $\mathbf{V}$ have been updated separately. They should be *synchronized* in order to let all nodes get the latest $\mathbf{U}$ and $\mathbf{V}$. It is obvious that during one iteration of processing all the ratings in the whole matrix, P synchronization operations should be performed for DSGD. This frequent synchronization will make DSGD inefficient because the slowest node will become the bottleneck of the whole system.

FPSGD [24], which is proposed for shared-memory systems, tries to improve the performance by changing the scheduler of DSGD into an asynchronous one. Its experiments show that FPSGD can achieve better efficiency and accuracy than DSGD.

Both Hogwild! and FPSGD are only for shared memory systems with one single node and thus their scalability is limited. DSGD can be used for distributed systems while it costs too much on synchronization.

2.5 ADMM

ADMM [3] is used to solve the constrained problems as follows:

$$\min_{\mathbf{x},\mathbf{z}} \quad f(\mathbf{x}) + g(\mathbf{z}) \quad (4)$$
$$s.t.: \quad \mathbf{A}\mathbf{x} + \mathbf{B}\mathbf{z} = \mathbf{c},$$

where $f(\cdot)$ and $g(\cdot)$ are functions, $\mathbf{x}$ and $\mathbf{z}$ are variables, $\mathbf{A}$, $\mathbf{B}$ and $\mathbf{c}$ are known values.

To solve the problem in (4), ADMM first gets the augmented Lagrangian as follows:

$$L(\mathbf{x}, \mathbf{z}, \mathbf{y}) = f(\mathbf{x}) + g(\mathbf{z}) + \mathbf{y}^T(\mathbf{A}\mathbf{x} + \mathbf{B}\mathbf{z} - \mathbf{c}) + \frac{\rho}{2}\|\mathbf{A}\mathbf{x} + \mathbf{B}\mathbf{z} - \mathbf{c}\|^2, \quad (5)$$

where $\mathbf{y}$ is the Lagrangian multiplier and ρ is the penalty parameter. The ADMM solution can be got be repeating the following three steps:

$$\mathbf{x}_{t+1} \leftarrow \operatorname*{argmin}_{\mathbf{x}} L(\mathbf{x}, \mathbf{z}_t, \mathbf{y}_t);$$
$$\mathbf{z}_{t+1} \leftarrow \operatorname*{argmin}_{\mathbf{z}} L(\mathbf{x}_{t+1}, \mathbf{z}, \mathbf{y}_t);$$
$$\mathbf{y}_{t+1} \leftarrow \mathbf{y}_t + \rho(\mathbf{A}\mathbf{x}_{t+1} + \mathbf{B}\mathbf{z}_{t+1} - \mathbf{c}),$$

where $\mathbf{x}_t$ denotes the value of $\mathbf{x}$ at the tth iteration, $\mathbf{y}_t$ and $\mathbf{z}_t$ are similarly defined. If $f(\mathbf{x})$ or $g(\mathbf{z})$ are separable, the corresponding steps of ADMM can be done in parallel. Hence, ADMM can be used to design distributed learning algorithms for large-scale problems [3].

In recent years, ADMM has captured more and more attention with wide applications, such as matrix completion [5], compressive sensing [19], image restoration [6] and response prediction [1]. Moreover, many variants of ADMM are also devised, including the stochastic and online extensions [15, 12, 17]. However, to the best of our knowledge, few works have been proposed to use stochastic ADMM for distributed MF problems.

3. DISTRIBUTED STOCHASTIC ADMM FOR MATRIX FACTORIZATION

In this section, we present the details of our DS-ADMM model. We first introduce our data split strategy to divide the whole problem into several sub-problems. Then we propose a distributed ADMM framework to handle these sub-problems in parallel. After that, a stochastic learning algorithm is designed to speed up the distributed ADMM framework. Subsequently, we compare the scheduler of DS-ADMM with those of DSGD and CCD++. Finally, the complexity analysis of DS-ADMM will be provided.

3.1 Data Split Strategy

In our data split strategy, we divide $\mathbf{R}$ and $\mathbf{U}$ into P sub-blocks according to users. More specifically, each sub-block will contain $\frac{m}{P}$ rows of $\mathbf{R}$ and $\frac{m}{P}$ columns of $\mathbf{U}$. From (1), we find that $\mathbf{U}$ and $\mathbf{V}$ are coupled together in the loss function.

Updating one of them needs the other's latest value, which makes the problem hardly separable. To decouple $\mathbf{U}$ and $\mathbf{V}$, we keep a local item latent matrix for all items in each node, which is denoted as $\mathbf{V}^p$. Please note that $\mathbf{V}^p$ is not a sub-block of $\mathbf{V}$, but it has the same size with $\mathbf{V}$. We also have a global item latent matrix which is denoted as $\overline{\mathbf{V}}$. Because only the local $\mathbf{V}^p$ couples with $\mathbf{U}$, we can independently update $\mathbf{U}$ and $\mathbf{V}^p$ for each node. This split strategy can make the MF problem fit for the distributed ADMM framework, which will be introduced in the following subsection.

Our split strategy is called *LocalMFSplit*, which is briefly summarized in Algorithm 1. Note that the size of $\mathbf{V}^p$ is $k \times n$, but that of $\mathbf{U}^p$ is $k \times m_p$ with m_p being the number of columns (about $\frac{m}{P}$) assigned to node p.

Algorithm 1 LocalMFSplit

1: **Input:** $\mathbf{R}$, P
2: **for** $i = 1 : m$ **do**
3: Generate a random number p from $\{1, 2, \cdots, P\}$, and distribute row i of $\mathbf{R}$ to node p.
4: **end for**
5: **for** $p = 1 : P$ parallel **do**
6: Allocate memory for $\mathbf{U}^p$, $\mathbf{V}^p$ and $\overline{\mathbf{V}}$
7: **end for**

3.2 Distributed ADMM

Based on our split strategy *LocalMFSplit*, the MF problem in (1) can be reformulated as follows:

$$\begin{aligned} \min_{\mathbf{U},\overline{\mathbf{V}},\mathcal{V}} \frac{1}{2} \sum_{p=1}^{P} \sum_{(i,j)\in\Omega^p} \Big[& (R_{i,j} - \mathbf{U}_{*i}^T \mathbf{V}_{*j}^p)^2 \\ & + \lambda_1 \mathbf{U}_{*i}^T \mathbf{U}_{*i} + \lambda_2 [\mathbf{V}_{*j}^p]^T \mathbf{V}_{*j}^p \Big] \\ s.t.: \quad & \mathbf{V}^p - \overline{\mathbf{V}} = 0; \quad \forall p \in \{1, 2, ..., P\} \end{aligned} \tag{6}$$

where $\mathcal{V} = \{\mathbf{V}^p\}_{p=1}^P$, Ω^p denotes the (i, j) indices of the ratings located in node p. Note that here we omit the p in $\mathbf{U}^p$ for simplicity. It is not hard to determine whether $\mathbf{U}$ refers to the whole latent matrix or a sub-block $\mathbf{U}^p$ located in node p from the context.

If we define

$$f(\mathbf{U}, \mathcal{V}) = \sum_{p=1}^{P} f^p(\mathbf{U}, \mathbf{V}^p), \tag{7}$$

where

$$\begin{aligned} f^p(\mathbf{U}, \mathbf{V}^p) &= \sum_{(i,j)\in\Omega^p} \hat{f}_{i,j}(\mathbf{U}_{*i}, \mathbf{V}_{*j}^p), \\ \hat{f}_{i,j}(\mathbf{U}_{*i}, \mathbf{V}_{*j}^p) &= \frac{1}{2} \Big[(R_{i,j} - \mathbf{U}_{*i}^T \mathbf{V}_{*j}^p)^2 \\ &\quad + \lambda_1 \mathbf{U}_{*i}^T \mathbf{U}_{*i} + \lambda_2 [\mathbf{V}_{*j}^p]^T \mathbf{V}_{*j}^p \Big], \end{aligned} \tag{8}$$

we can transform the constrained problem in (6) to an unconstrained problem with augmented Lagrangian method, and get the following objective function:

$$L(\mathbf{U}, \mathcal{V}, \mathcal{O}, \overline{\mathbf{V}}) = f(\mathbf{U}, \mathcal{V}) + l(\mathcal{V}, \overline{\mathbf{V}}, \mathcal{O}), \tag{9}$$

where

$$\begin{aligned} l(\mathcal{V}, \overline{\mathbf{V}}, \mathcal{O}) &= \sum_{p=1}^{P} l^p(\mathbf{V}^p, \overline{\mathbf{V}}, \mathbf{\Theta}^p), \\ l^p(\mathbf{V}^p, \overline{\mathbf{V}}, \mathbf{\Theta}^p) &= \Big[\frac{\rho}{2} \|\mathbf{V}^p - \overline{\mathbf{V}}\|_F^2 + \mathrm{tr}\big([\mathbf{\Theta}^p]^T (\mathbf{V}^p - \overline{\mathbf{V}})\big) \Big]. \end{aligned}$$

Here, ρ is a hyper-parameter and $\mathcal{O} = \{\mathbf{\Theta}^p\}_{p=1}^P$ denotes the Lagrangian multiplier.

If we define

$$\begin{aligned} L^p(\mathbf{U}, \mathbf{V}^p, \mathbf{\Theta}^p, \overline{\mathbf{V}}) &= f^p(\mathbf{U}, \mathbf{V}^p) + l^p(\mathbf{V}^p, \overline{\mathbf{V}}, \mathbf{\Theta}^p) \\ &= \sum_{(i,j)\in\Omega^p} \hat{f}_{i,j}(\mathbf{U}_{*i}, \mathbf{V}_{*j}^p) \\ &\quad + \Big[\frac{\rho}{2} \|\mathbf{V}^p - \overline{\mathbf{V}}\|_F^2 + \mathrm{tr}\big([\mathbf{\Theta}^p]^T (\mathbf{V}^p - \overline{\mathbf{V}})\big) \Big], \end{aligned}$$

we can get

$$L(\mathbf{U}, \mathcal{V}, \mathcal{O}, \overline{\mathbf{V}}) = \sum_{p=1}^{P} L^p(\mathbf{U}, \mathbf{V}^p, \mathbf{\Theta}^p, \overline{\mathbf{V}}).$$

The ADMM will solve this problem by repeating the following steps:

$$\mathbf{U}_{t+1}, \mathbf{V}_{t+1}^p \leftarrow \operatorname*{argmin}_{\mathbf{U},\mathbf{V}^p} L^p(\mathbf{U}, \mathbf{V}^p, \mathbf{\Theta}_t^p, \overline{\mathbf{V}}_t), \forall p \in \{1, 2, ..., P\} \tag{10a}$$

$$\overline{\mathbf{V}}_{t+1} \leftarrow \operatorname*{argmin}_{\overline{\mathbf{V}}} L(\mathbf{U}_{t+1}, \mathcal{V}_{t+1}, \mathcal{O}_t, \overline{\mathbf{V}}), \tag{10b}$$

$$\mathbf{\Theta}_{t+1}^p \leftarrow \mathbf{\Theta}_t^p + \rho(\mathbf{V}_{t+1}^p - \overline{\mathbf{V}}_{t+1}), \forall p \in \{1, 2, ..., P\}. \tag{10c}$$

It is easy to see that $\mathbf{U}, \mathbf{V}^p$ and $\mathbf{\Theta}^p$ can be locally updated on each node. Because the whole MF problem has been divided into P sub-problems which can be solved in parallel, our method is actually a distributed ADMM framework.

3.3 Stochastic Learning for Distributed ADMM

To learn the parameters in (6), we just need to find the solutions in (10a), (10b) and (10c). After getting the optimal $\mathbf{U}_{t+1}$ and $\{\mathbf{V}_{t+1}^p\}$, it is easy to solve the problem in (10b). More specifically, if we set $\mathbf{\Theta}_0^p = 0$, we can prove that $\sum_{p=1}^P \mathbf{\Theta}_t^p = 0$. Hence, the update rule for $\overline{\mathbf{V}}$ is:

$$\overline{\mathbf{V}}_{t+1} = \frac{1}{P} \sum_{p=1}^{P} \mathbf{V}_{t+1}^p. \tag{11}$$

The problem in (10c) directly shows the update rule, which can be computed locally and efficiently. Therefore, the key learning part lies in how to efficiently solve the problem in (10a). In the following content of this subsection, we first design a batch learning algorithm for the problem in (10a), and then a stochastic learning algorithm inspired by the batch learning is also designed to further improve the efficiency.

3.3.1 Batch Learning

With $\mathbf{\Theta}_t^p$ and $\overline{\mathbf{V}}_t$ fixed, (10a) is an MF problem. However, we can not easily get the solution because $\mathbf{U}$ and $\mathbf{V}^p$ are coupled together and the objective function of the MF problem is non-convex. To get an efficient solution, we use a technique similar to that in [12] to construct a surrogate

objective function, which is convex and can make $\mathbf{U}$ and $\mathbf{V}$ decouple from each other. For each iteration of minimizing the constructed function, we can easily get the closed form solution of $\mathbf{U}$ and $\mathbf{V}^p$ by setting their gradients to zero.

The surrogate objective function is defined as follows:

$$G^p(\mathbf{U},\mathbf{V}^p,\mathbf{\Theta}_t^p,\overline{\mathbf{V}}_t,\tau_t|\mathbf{U}_t,\mathbf{V}_t^p) = g^p(\mathbf{U},\mathbf{V}^p,\tau_t|\mathbf{U}_t,\mathbf{V}_t^p) + l^p(\mathbf{V}^p,\overline{\mathbf{V}}_t,\mathbf{\Theta}_t^p), \quad (12)$$

where

$$\begin{aligned} g^p(\mathbf{U},\mathbf{V}^p,\tau_t|\mathbf{U}_t,\mathbf{V}_t^p) &= f^p(\mathbf{U}_t,\mathbf{V}_t^p) + \mathrm{tr}[\nabla_{\mathbf{U}}^T f^p(\mathbf{U}_t,\mathbf{V}_t^p)(\mathbf{U}-\mathbf{U}_t)] \\ &+ \mathrm{tr}[\nabla_{\mathbf{V}^p}^T f^p(\mathbf{U}_t,\mathbf{V}_t^p)(\mathbf{V}^p-\mathbf{V}_t^p)] \\ &+ \frac{1}{2\tau_t}(\|\mathbf{U}-\mathbf{U}_t\|_F^2 + \|\mathbf{V}^p-\mathbf{V}_t^p\|_F^2), \end{aligned} \quad (13)$$

with τ_t being a value which will be useful for specifying the step-size in the stochastic learning method introduced later, and the function $f^p(\mathbf{U},\mathbf{V}^p)$ being defined in (8).

LEMMA 1. *For an arbitrary positive value δ^2, we can always find a τ_t that makes $G^p(\cdot)$ satisfy the following two properties within the domain $D = \{\mathbf{U},\mathbf{V}^p \mid \|\mathbf{U}_{*i}-[\mathbf{U}_{*i}]_t\|_F^2 \le \delta^2, \|\mathbf{V}_{*j}^p - [\mathbf{V}_{*j}^p]_t\|_F^2 \le \delta^2\}$:*

$$G^p(\mathbf{U},\mathbf{V}^p,\mathbf{\Theta}_t^p,\overline{\mathbf{V}}_t,\tau_t|\mathbf{U}_t,\mathbf{V}_t^p) \ge L^p(\mathbf{U},\mathbf{V}^p,\mathbf{\Theta}_t^p,\overline{\mathbf{V}}_t),$$
$$G^p(\mathbf{U}_t,\mathbf{V}_t^p,\mathbf{\Theta}_t^p,\overline{\mathbf{V}}_t,\tau_t|\mathbf{U}_t,\mathbf{V}_t^p) = L^p(\mathbf{U}_t,\mathbf{V}_t^p,\mathbf{\Theta}_t^p,\overline{\mathbf{V}}_t).$$

The proof of Lemma 1 can be found in Appendix A.

From Lemma 1 , we can find that $G^p(\cdot)$ is an upper bound of $L^p(\cdot)$, and $G^p(\cdot) = L^p(\cdot)$ at the point $(\mathbf{U}_t,\mathbf{V}_t^p)$. Compared with $L^p(\cdot)$, $\mathbf{U}$ and $\mathbf{V}^p$ are decoupled in $G^p(\cdot)$, and $G^p(\cdot)$ is convex in $(\mathbf{U},\mathbf{V}^p)$. Hence, it is much easier to optimize $G^p(\cdot)$ than $L^p(\cdot)$.

Instead of optimizing the original function $L^p(\cdot)$, we optimize the surrogate function $G^p(\cdot)$ in the first step of the ADMM:

$$\mathbf{U}_{t+1},\mathbf{V}_{t+1}^p \leftarrow \underset{\mathbf{U},\mathbf{V}^p}{\mathrm{argmin}}\, G^p(\mathbf{U},\mathbf{V}^p,\mathbf{\Theta}_t^p,\overline{\mathbf{V}}_t,\tau_t|\mathbf{U}_t,\mathbf{V}_t^p) \quad (14a)$$

The objective function in (14a) is convex in both $\mathbf{U}$ and $\mathbf{V}^p$. Hence, we can easily get the solution by setting the gradients to be zero. The optimal solution is computed as follows:

$$\mathbf{U}_{t+1} = \mathbf{U}_t - \tau_t * \nabla_{\mathbf{U}}^T f^p(\mathbf{U}_t,\mathbf{V}_t^p), \quad (15)$$

$$\mathbf{V}_{t+1}^p = \frac{\tau_t}{1+\rho\tau_t}[\frac{\mathbf{V}_t^p}{\tau_t} + \rho\overline{\mathbf{V}}_t - \mathbf{\Theta}_t^p - \nabla_{\mathbf{V}^p}^T f^p(\mathbf{U}_t,\mathbf{V}_t^p)]. \quad (16)$$

LEMMA 2. *By following the update rules in* (15) *and* (16)*, the original objective function $L^p(\cdot)$ will not increase in each step. That is to say,*

$$L^p(\mathbf{U}_{t+1},\mathbf{V}_{t+1}^p,\mathbf{\Theta}_t^p,\overline{\mathbf{V}}_t) \le L^p(\mathbf{U}_t,\mathbf{V}_t^p,\mathbf{\Theta}_t^p,\overline{\mathbf{V}}_t). \quad (17)$$

The proof of Lemma 2 can be found in Appendix B.

By combining the update rules in (15), (16), (11) and (10c), we can get a batch learning algorithm for the problem in (6) with the distributed ADMM framework.

THEOREM 1. *Our batch learning algorithm will converge.*

PROOF. Based on Lemma 2, we can prove that the objective function $L(\cdot)$ in (9) will decrease in each iteration of ADMM. Furthermore, $L(\cdot)$ is lower bounded by $-\frac{\sum_{p=1}^P ||\mathbf{\Theta}^p||_F^2}{2\rho}$. Hence, our batch learning algorithm will converge. Because $L(\cdot)$ is not convex, it might converge to a local minimum. □

3.3.2 Stochastic Learning

From (15), we can find that it will access all ratings related to $\mathbf{U}_{*i}$ to update each $\mathbf{U}_{*i}$, and the same also goes for updating each $\mathbf{V}_{*j}^p$ in (16). Hence, the batch learning algorithm presented above is not efficient, especially when the number of ratings becomes very large. To further improve the efficiency, we propose a stochastic learning strategy for the distributed ADMM, which is called DS-ADMM. In particular, the update rules for DS-ADMM is as follows:

$$(\mathbf{U}_{*i})_{t+1} = (\mathbf{U}_{*i})_t + \tau_t(\epsilon_{ij}(\mathbf{V}_{*j}^p)_t - \lambda_1(\mathbf{U}_{*i})_t), \quad (18)$$

$$(\mathbf{V}_{*j}^p)_{t+1} = \frac{\tau_t}{1+\rho\tau_t}[\frac{1-\lambda_2\tau_t}{\tau_t}(\mathbf{V}_{*j}^p)_t + \epsilon_{ij}(\mathbf{U}_{*i})_t + \rho(\overline{\mathbf{V}}_{*j})_t - (\mathbf{\Theta}_{*j}^p)_t], \quad (19)$$

where $\epsilon_{ij} = R_{ij} - [(\mathbf{U}_{*i})_t]^T(\mathbf{V}_{*j}^p)_t$. It is easy to see that the stochastic learning algorithm is derived from the batch learning algorithm by treating only $\mathbf{U}_{*i}$ and $\mathbf{V}_{*j}^p$ as variables in (14a).

By combining the split strategy and the update rules stated above, we can get our DS-ADMM algorithm. The whole procedure of DS-ADMM is briefly listed in Algorithm 2.

Algorithm 2 DS-ADMM

```
1: Input: R, P, ρ, MaxIter, λ1, λ2, τ0;
2: Use Algorithm 1 to distribute R to P different nodes.
3: Randomly initialize U0, V0^p;
4: Calculate V̄0 by (11)
5: Set Θ0^p = 0.
6: for t = 1 : MaxIter do
7:    for p = 1 : P parallel do
8:       for each R_{i,j} in node p do
9:          Update U_{*i} and V_{*j}^p by (18) and (19)
10:      end for
11:   end for
12:   Update V̄ by (11)
13:   for p = 1 : P parallel do
14:      Update Θ^p by (10c)
15:   end for
16:   Update τ_t
17: end for
```

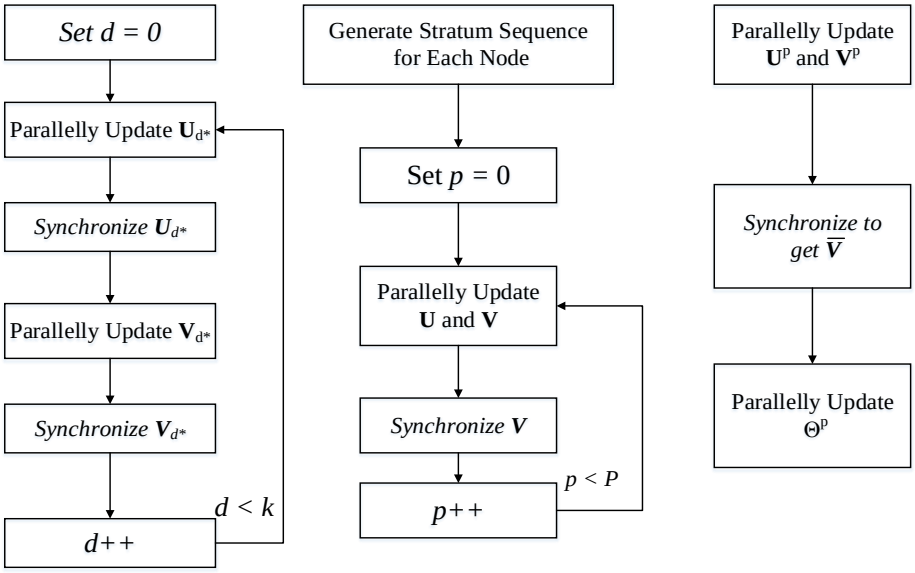
3.4 Scheduler Comparison

CCD++ and DSGD are two state-of-the-art distributed MF models. We compare the scheduler of our DS-ADMM with those of CCD++ and DSGD to illustrate the synchronization cost.

Figure 1 (a), (b) and (c) show the number of synchronization operations in one iteration of CCD++, DSGD and DS-ADMM, respectively. Here, one iteration means all the training ratings are processed for one time. We can find that CCD++ needs $2k$ times of synchronization and DSGD needs P times. From Algorithm 2, we can easily find that DS-ADMM needs only one synchronization for each iteration, which is shown in line 12. This synchronization step is used to gather all $\mathbf{V}^p$. Hence, it is obvious that the synchronization cost of our DS-ADMM is much less than those of CCD++ and DSGD.

3.5 Complexity Analysis

DS-ADMM updates all variables once by three steps. Step one updates $\mathbf{U}$ and $\mathbf{V}^p$. For each rating R_{ij}, the time com-



(a) CCD++ (b) DSGD (c) DS-ADMM

Figure 1: Operations in one iteration of CCD++, DSGD, and DS-ADMM. It shows that CCD++ and DSGD need multiple times of synchronization while DS-ADMM needs only one synchronization for each iteration.

plexity to update $\mathbf{U}_{*i}$ and $\mathbf{V}^p_{*j}$ is $O(k^2)$. Because the total number of observed ratings is $|\Omega|$, the time complexity of step one is $O(k^2|\Omega|)$. Step two is actually a summation of P matrices of size $k \times n$, thus the time complexity is $O(Pnk)$. Step three need to update P matrices of size $k \times n$, and the update operation only contains constant times of addition, so the time complexity is $O(Pnk)$. In total, the time complexity of DS-ADMM for each iteration is $O(k^2|\Omega| + Pnk)$.

4. EXPERIMENTS

All the experiments are run on an MPI-cluster with twenty nodes, each of which is a 24-core server with 2.2GHz Intel(R) Xeon(R) E5-2430 processor and 96GB of RAM. To evaluate the scale-out performance of our model, we use only one 1 core (thread) and 10GB memory for each node.

4.1 Data Sets

We run our experiments on three public collaborative filtering data sets: Netflix [2], Yahoo! Music R1, and Yahoo! Music R2 [3]. The Netflix data set contains the users' ratings to movies. Yahoo! Music R1 data set contains the users' ratings to artists. Yahoo! Music R2 contains the users' ratings to songs.

As the original ratings of Yahoo! Music R1 are ranging from $0 - 100$ and have value *Never play again*, we treat the ratings with value 0 and *Never play again* as the explicit negative feedback and filter them out. For the other ratings, we normalize them by multiplying each rating by 0.05. After preprocessing, all the ratings lie in the range of $[0.05, 5]$.

The Netflix and Yahoo! Music R2 data sets also contain public test data sets, which are used in our experiments. For the Yahoo! Music R1 data set, we randomly select 10% of the ratings for testing and the remaining are used for training. Detailed information of these data sets are shown in the first four rows in Table 1.

Table 1: Data sets and parameter settings

Data Set	Netflix	Yahoo! Music R1	Yahoo! Music R2
m	480,190	1,938,361	1,823,179
n	17,770	49,995	136,736
#Train	99,072,112	73,578,902	699,640,226
#Test	1,408,395	7,534,592	18,231,790
k	40	40	40
η_0/τ_0	0.1	0.1	0.1
λ_1	0.05	0.05	0.05
λ_2	0.05	0.05	0.05
ρ	0.05	0.05	0.1
α	0.002	0.002	0.006
β	0.7	0.7	0.7
P	8	10	20

4.2 Baselines and Parameter Settings

FPSGD and Hogwild! can only run on multi-core systems while we focus on distributed algorithms in this paper. So CCD++, DSGD and DSGD-Bias are adopted as our baselines.

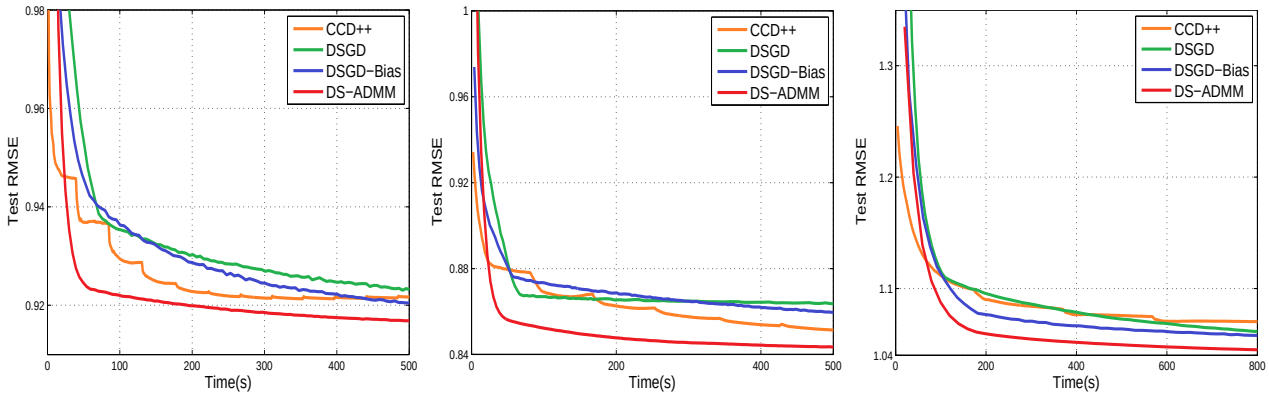
CCD++ is implemented by referring to the public OpenMP version[4]. DSGD is implemented according to [4] with an asynchronous scheduler to improve its performance.

We also implement a variant of DSGD called DSGD-Bias by using the prediction function: $R_{i,j} = \mu + b_i + b_j + \mathbf{U}^T_{*i}\mathbf{V}_{*j}$, where b_i, b_j are the user and item bias for ratings and μ is the global mean of the ratings. The model with bias is an

[2]http://www.netflixprize.com/

[3]http://webscope.sandbox.yahoo.com/catalog.php?datatype=r

[4]http://www.cs.utexas.edu/∼rofuyu/libpmf/



(a) Netflix (b) Yahoo!Music R1 (c) Yahoo!Music R2

Figure 2: Test RMSE curve for CCD++, DSGD, DSGD-Bias and DS-ADMM. (a) Netflix data set; (b)Yahoo! Music R1 data set; (c) Yahoo! Music R2 data set. The vertical axis is the RMSE on the test set, and the horizontal axis is the running time.

easy and natural extension of MF and has been proved to be more accurate than those without bias [8]. DSGD-Bias model is optimized and paralleled by using the same strategy as that in DSGD.

During our experiments, we find that good results can be achieved by setting $\lambda_1 = \lambda_2$ for all the algorithms. So we simply set $\lambda_1 = \lambda_2$ in our experiments. All the hyper-parameters for each model in our experiments are selected by ten-fold cross validation on the training set except the latent factor number k and the number of computing nodes P. k is selected by following the CCD++ [21]. Actually, we find that on our data sets larger k doesn't achieve much better accuracy for CCD++ while increasing the running time. The node number P is set according to the size of the data sets. All the algorithms have the same P for the same data set.

We use a general and simple update rule for DSGD's learning rate. More specifically, we first initialize the learning rate η with a relatively large value, then decrease it by $\eta_{t+1} = \eta_t * \beta$ $(0 < \beta < 1)$ after each iteration, and stop decreasing when η becomes smaller than some threshold α. This strategy results in a fast convergence rate and avoids early stopping.

DS-ADMM has a similar step-size parameter τ_t. From the detailed proof in Appendix A, we find that τ_t should be smaller than some threshold computed based on $\mathbf{U}_t$ and $\mathbf{V}_t^p$. Because the exact threshold value for τ_t is hard to calculate, we approximately update it as $\tau_{t+1} = \tau_t * \beta$ $(0 < \beta < 1)$ for the tth iteration. We also set a threshold α. When $\tau_t \leq \alpha$, we stop decreasing τ_t. It is easy to find that the update rule for τ_t is the same as that for the DSGD's learning rate η.

The parameter settings are shown in the last eight rows in Table 1.

4.3 Accuracy and Efficiency

The root mean squared error (RMSE) is a widely used metric to measure an MF model's performance [21]. The test RMSE is defined as: $\frac{1}{Q}\sqrt{\sum (R_{i,j} - \mathbf{U}_{*i}^T \overline{\mathbf{V}}_{*j})^2}$, where Q is the number of testing ratings. Figure 2 shows the test RMSE versus running time for our DS-ADMM and other baselines. We can easily find that DS-ADMM performs the best on all three data sets. RMSE value of CCD++ and the DSGD-Bias model decreases fast at first for some data sets because CCD++ updates the parameters by their closed-form solutions and the DSGD-Bias model extracts more explicit information from the training data set. However, our DS-ADMM decreases much faster afterwards and finally converges to the best accuracy, which is much better than the DSGD-Bias model and CCD++. DSGD performs the worst among all the models in most cases. Moreover, as the scale of the data set grows, the difference between CCD++, DSGD and DSGD-bias's convergence value becomes smaller, but their difference to DS-ADMM becomes larger. Note that Yahoo! Music R2 has about 0.7 billion ratings, which is much larger than many real world applications.

In some application scenarios, the learning process stops when the RMSE value reaches some threshold. So we develop experiments to test those algorithms' running time to reach the given threshold with different number of computing nodes. When conducting experiments on the Netflix data set, we set the threshold of RMSE value as 0.922. This value is chosen because it is the place where the RMSE curves of DS-ADMM and CCD++ become smooth, and it is also a value that DSGD and DSGD-Bias can reach if provided with enough running time. We report the log-scale of the running time in Figure 3. Results on the other two data sets are similar, which are omitted due to the limited space. From Figure 3, we can find that our DS-ADMM outperforms all the other algorithms no matter how many nodes are used. DS-ADMM needs relatively fewer iterations to reach a test RMSE of 0.922 and its running time for each iteration is the smallest among all algorithms. DSGD performs the worst since it should run more iterations to reach a test RMSE of 0.922 and the running time for each iteration is also relatively large.

4.4 Speedup

Another important metric that can be used to measure a distributed algorithm is its speedup or scalability. It measures the performance when more machines are used or larger data sets are processed. In general, more machines and

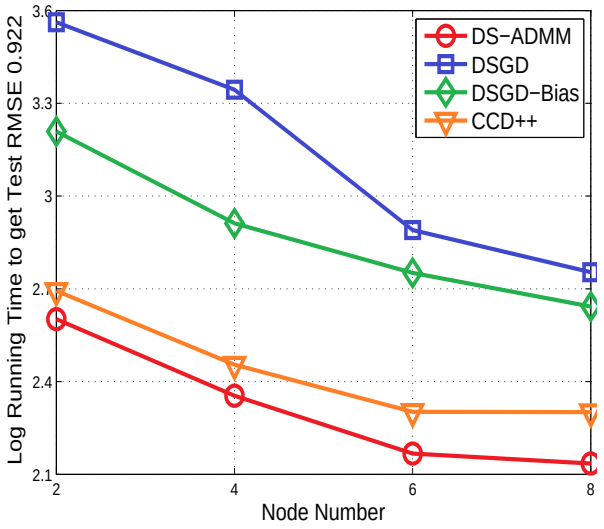


Figure 3: Average running time to reach a test RMSE of 0.922 on the Netflix data set for different methods.

larger data sets will increase the communication and scheduling costs. Algorithms with poor scalability consume more computing resources without offering a good reward, so they are not suitable for large-scale applications.

To study the scalability of our algorithm, we compute the speedup factor relative to the running time with 2 nodes by varying the number of nodes from 2 to 8. Speedup factors of CCD++ and DSGD are provided for comparison. The results on the Netflix data set are shown in Figure 4. Results on other data sets are similar, which are omitted for space saving.

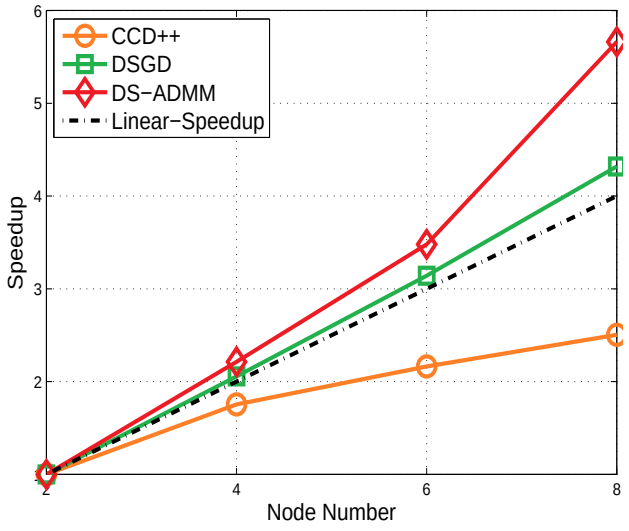


Figure 4: Speedup comparison between CCD++, DSGD, and DS-ADMM on the Netflix data set.

It is obvious from Figure 4 that DS-ADMM achieves the best speedup among all three algorithms. Figure 4 also shows that both DSGD and DS-ADMM have a super-linear speedup. This might be reasonable. More specifically, as the number of computing nodes increases, the scale of the data set on each node decreases. With a relatively large cache size, most data or even the whole data set can be loaded into caches and thus the accessing cost is reduced, which will gain extra speedup [2, 7][5].

By separately measuring the computing cost and synchronization cost in our experiments, we find that DSGD costs much more time on communication and synchronization than DS-ADMM. Taking the Netflix experiment as an example, although the total running time for each iteration are nearly the same for DSGD and DS-ADMM, DSGD spends about 18% of the total time on communication and synchronization, while that is only 8% for DS-ADMM. This result verifies our previous analysis on the synchronization issue in DSGD.

4.5 Sensitivity to Hyper-parameter ρ

We vary the values of ρ to study its effect on the performance of DS-ADMM, the results of which are shown in Figure 5. We can find that very good performance can be achieved when ρ is around 0.05 for all data sets, and our DS-ADMM is not sensitive to ρ in the range [0.03, 0.1]. This relatively stable property of ρ makes hyper-parameter selection much easier.

5. CONCLUSION

In this paper, we propose a new distributed algorithm called DS-ADMM for large-scale matrix factorization in recommender systems. We first design a data split strategy to divide the large-scale problem into several sub-problems and then derive a distributed stochastic variant of ADMM for efficient learning. Experiments on real world data sets show that our DS-ADMM algorithm can outperform the state-of-the-art methods like DSGD and CCD++ in terms of accuracy, efficiency and scalability.

6. ACKNOWLEDGEMENTS

This work is supported by the NSFC (No. 61100125, No. 61472182), the 863 Program of China (No. 2012AA011003), and the Program for Changjiang Scholars and Innovative Research Team in University of China (IRT1158, PCSIRT).

7. REFERENCES

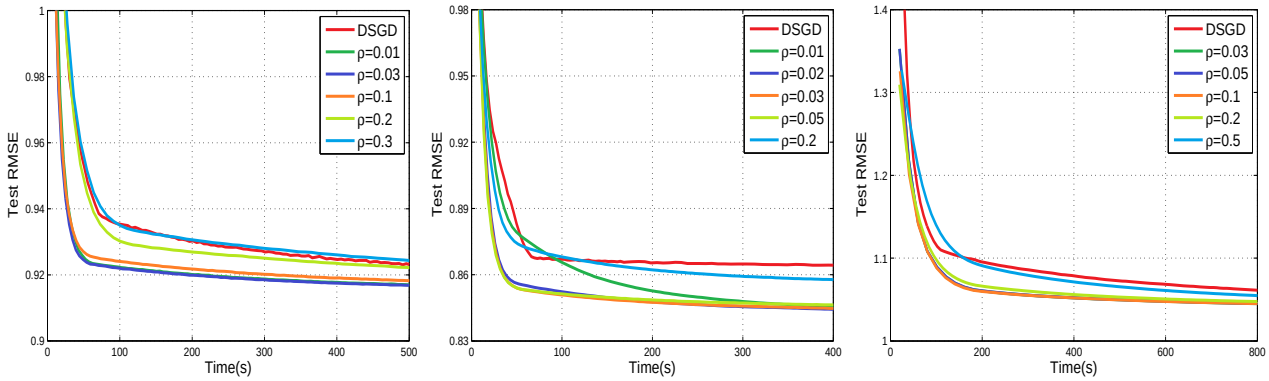
[1] D. Agarwal. Computational advertising: the linkedin way. In *CIKM*, pages 1585–1586, 2013.

[2] J. Benzi and M. Damodaran. Parallel three dimensional direct simulation monte carlo for simulating micro flows. In *Parallel Computational Fluid Dynamics 2007*, pages 91–98. Springer, 2009.

[3] S. P. Boyd, N. Parikh, E. Chu, B. Peleato, and J. Eckstein. Distributed optimization and statistical learning via the alternating direction method of multipliers. *Foundations and Trends in Machine Learning*, pages 1–122, 2011.

[4] R. Gemulla, E. Nijkamp, P. J. Haas, and Y. Sismanis. Large-scale matrix factorization with distributed stochastic gradient descent. In *KDD*, pages 69–77, 2011.

[5] `http://en.wikipedia.org/wiki/Speedup`



(a) Netflix (b) Yahoo!Music R1 (c) Yahoo!Music R2

Figure 5: The effect of ρ in DS-ADMM on three data sets.

[5] D. Goldfarb, S. Ma, and K. Scheinberg. Fast alternating linearization methods for minimizing the sum of two convex functions. *Math. Program.*, pages 349–382, 2013.

[6] T. Goldstein and S. Osher. The split bregman method for l1-regularized problems. *SIAM J. Imaging Sciences*, pages 323–343, 2009.

[7] B. John, X.-J. Gu, and D. R. Emerson. Investigation of heat and mass transfer in a lid-driven cavity under nonequilibrium flow conditions. *Numerical Heat Transfer, Part B: Fundamentals*, 58(5):287–303, 2010.

[8] Y. Koren, R. M. Bell, and C. Volinsky. Matrix factorization techniques for recommender systems. *IEEE Computer*, 42(8):30–37, 2009.

[9] W.-J. Li and D.-Y. Yeung. Relation regularized matrix factorization. In *IJCAI*, pages 1126–1131, 2009.

[10] Y. Li, M. Yang, and Z. M. Zhang. Scientific articles recommendation. In *CIKM*, pages 1147–1156, 2013.

[11] F. Niu, B. Recht, C. Re, and S. J. Wright. Hogwild!: A lock-free approach to parallelizing stochastic gradient descent. In *NIPS*, pages 693–701, 2011.

[12] H. Ouyang, N. He, L. Tran, and A. G. Gray. Stochastic alternating direction method of multipliers. In *ICML*, pages 80–88, 2013.

[13] I. Pilěřl'szy, D. Zibriczky, and D. Tikk. Fast als-based matrix factorization for explicit and implicit feedback datasets. In *RecSys*, pages 71–78, 2010.

[14] S. Purushotham and Y. Liu. Collaborative topic regression with social matrix factorization for recommendation systems. In *ICML*, 2012.

[15] T. Suzuki. Dual averaging and proximal gradient descent for online alternating direction multiplier method. In *ICML*, pages 392–400, 2013.

[16] C. Wang and D. M. Blei. Collaborative topic modeling for recommending scientific articles. In *KDD*, pages 448–456, 2011.

[17] H. Wang and A. Banerjee. Online alternating direction method. In *ICML*, 2012.

[18] H. Wang, B. Chen, and W.-J. Li. Collaborative topic regression with social regularization for tag recommendation. In *IJCAI*, 2013.

[19] J. Yang and Y. Zhang. Alternating direction algorithms for problems in compressive sensing. *SIAM J. Scientific Computing*, pages 250–278, 2011.

[20] T. Yang. Trading computation for communication: Distributed stochastic dual coordinate ascent. In *NIPS*, pages 629–637, 2013.

[21] H.-F. Yu, C.-J. Hsieh, S. Si, and I. S. Dhillon. Scalable coordinate descent approaches to parallel matrix factorization for recommender systems. In *ICDM*, pages 765–774, 2012.

[22] Y. Zhen, W.-J. Li, and D.-Y. Yeung. Tagicofi: tag informed collaborative filtering. In *RecSys*, pages 69–76, 2009.

[23] Y. Zhou, D. M. Wilkinson, R. Schreiber, and R. Pan. Large-scale parallel collaborative filtering for the netflix prize. In *AAIM*, pages 337–348, 2008.

[24] Y. Zhuang, W.-S. Chin, Y.-C. Juan, and C.-J. Lin. A fast parallel sgd for matrix factorization in shared memory systems. In *RecSys*, pages 249–256, 2013.

[25] M. Zinkevich, M. Weimer, A. J. Smola, and L. Li. Parallelized stochastic gradient descent. In *NIPS*, pages 2595–2603, 2010.

APPENDIX

A. PROOF OF LEMMA 1

PROOF. We can rewrite

$$g^p(\mathbf{U},\mathbf{V}^p,\tau_t|\mathbf{U}_t,\mathbf{V}_t^p)=\sum_{(i,j)\in\Omega^p} g_{i,j}^p(\mathbf{U}_{*i},\mathbf{V}_{*j}^p,\tau_t|\mathbf{U}_t,\mathbf{V}_t^p),$$

where

$$\begin{aligned} g_{i,j}^p(\mathbf{U}_{*i},\mathbf{V}_{*j}^p,&\tau_t|\mathbf{U}_t,\mathbf{V}_t^p)\\ &=\Big[\hat{f}_{i,j}([\mathbf{U}_{*i}]_t,[\mathbf{V}_{*j}^p]_t)\\ &+\nabla_{\mathbf{U}_{*i}}^T\hat{f}_{i,j}([\mathbf{U}_{*i}]_t,[\mathbf{V}_{*j}^p]_t)(\mathbf{U}_{*i}-[\mathbf{U}_{*i}]_t)\\ &+\nabla_{\mathbf{V}_{*j}^p}^T\hat{f}_{i,j}([\mathbf{U}_{*i}]_t,[\mathbf{V}_{*j}^p]_t)(\mathbf{V}_{*j}^p-[\mathbf{V}_{*j}^p]_t)\\ &+\frac{1}{2m_i\tau_t}\|\mathbf{U}_{*i}-[\mathbf{U}_{*i}]_t\|_F^2\\ &+\frac{1}{2n_j\tau_t}\|\mathbf{V}_{*j}^p-[\mathbf{V}_{*j}^p]_t\|_F^2\Big], \end{aligned}$$

with $m_i = |\Omega_i^p|$ and $n_j = |\tilde{\Omega}_j^p|$.

Then, we have

$$\begin{aligned}
&L^p(\mathbf{U}, \mathbf{V}^p, \mathbf{\Theta}_t^p, \overline{\mathbf{V}}_t) - G^p(\mathbf{U}, \mathbf{V}^p, \mathbf{\Theta}_t^p, \overline{\mathbf{V}}_t, \tau_t | \mathbf{U}_t, \mathbf{V}_t^p) \\
&= f^p(\mathbf{U}, \mathbf{V}^p) - g^p(\mathbf{U}, \mathbf{V}^p, \tau_t | \mathbf{U}_t, \mathbf{V}_t^p) \\
&= \sum_{(i,j)\in\Omega^p} \left[\hat{f}_{i,j}(\mathbf{U}_{*i}, \mathbf{V}_{*j}^p) - g_{i,j}^p(\mathbf{U}_{*i}, \mathbf{V}_{*j}^p, \tau_t | \mathbf{U}_t, \mathbf{V}_t^p) \right],
\end{aligned}$$

For clarity, we denote $\mathbf{U}_{*i}$,$\mathbf{V}_{*j}^p$,$[\mathbf{U}_{*i}]_t$ and $[\mathbf{V}_{*j}^p]_t$ as $\mathbf{u}$,$\mathbf{v}$,$\mathbf{u}_t$ and $\mathbf{v}_t$, respectively. Then we have

$$\begin{aligned}
\hat{f}_{i,j}(\mathbf{u}, \mathbf{v}) = \frac{1}{2} \Big[&\big(R_{i,j} - (\mathbf{u} - \mathbf{u}_t + \mathbf{u}_t)^T (\mathbf{v} - \mathbf{v}_t + \mathbf{v}_t)\big)^2 \\
&+ \lambda_1 \|\mathbf{u} - \mathbf{u}_t + \mathbf{u}_t\|_F^2 + \lambda_2 \|\mathbf{v} - \mathbf{v}_t + \mathbf{v}_t\|_F^2 \Big] \\
= \hat{f}_{i,j}(\mathbf{u}_t, \mathbf{v}_t) &+ \nabla_{\mathbf{u}}^T \hat{f}_{i,j}(\mathbf{u}_t, \mathbf{v}_t)(\mathbf{u} - \mathbf{u}_t) \\
&+ \nabla_{\mathbf{v}}^T \hat{f}_{i,j}(\mathbf{u}_t, \mathbf{v}_t)(\mathbf{v} - \mathbf{v}_t) \\
&+ h(\mathbf{u}, \mathbf{v}) + o(\mathbf{u}, \mathbf{v}),
\end{aligned}$$

where

$$\begin{aligned}
\nabla_{\mathbf{u}}^T \hat{f}_{i,j}(\mathbf{u}_t, \mathbf{v}_t) &= \lambda_1 \mathbf{u}_t - (R_{i,j} - \mathbf{u}_t^T \mathbf{v}_t)\mathbf{v}_t, \\
\nabla_{\mathbf{v}}^T \hat{f}_{i,j}(\mathbf{u}_t, \mathbf{v}_t) &= \lambda_2 \mathbf{v}_t - (R_{i,j} - \mathbf{u}_t^T \mathbf{v}_t)\mathbf{u}_t,
\end{aligned}$$

$h(\mathbf{u}, \mathbf{v})$ contains all the second order terms, and the third order and forth order terms are contained in $o(\mathbf{u}, \mathbf{v})$. Hence, we can get

$$\begin{aligned}
h(\mathbf{u}, \mathbf{v}) = &\frac{\lambda_1}{2} \|\mathbf{u} - \mathbf{u}_t\|_F^2 + \frac{\lambda_2}{2} \|\mathbf{v} - \mathbf{v}_t\|_F^2 \\
&- (R_{i,j} - \mathbf{u}_t^T \mathbf{v}_t)(\mathbf{u} - \mathbf{u}_t)^T (\mathbf{v} - \mathbf{v}_t) \\
&+ \frac{1}{2} \big(\mathbf{u}_t^T (\mathbf{v} - \mathbf{v}_t) + (\mathbf{u} - \mathbf{u}_t)^T \mathbf{v}_t\big)^2,
\end{aligned}$$

and

$$\begin{aligned}
o(\mathbf{u}, \mathbf{v}) = &\frac{1}{2} [2(\mathbf{u} - \mathbf{u}_t)^T \mathbf{v}_t + 2(\mathbf{v} - \mathbf{v}_t)^T \mathbf{u}_t \\
&+ (\mathbf{u} - \mathbf{u}_t)^T (\mathbf{v} - \mathbf{v}_t)](\mathbf{u} - \mathbf{u}_t)^T (\mathbf{v} - \mathbf{v}_t).
\end{aligned}$$

Because

$$\begin{aligned}
&- (R_{i,j} - \mathbf{u}_t^T \mathbf{v}_t)(\mathbf{u} - \mathbf{u}_t)^T (\mathbf{v} - \mathbf{v}_t) \\
&\le \frac{1}{2} |R_{i,j} - \mathbf{u}_t^T \mathbf{v}_t| \big(\|\mathbf{u} - \mathbf{u}_t\|_F^2 + \|\mathbf{v} - \mathbf{v}_t\|_F^2\big),
\end{aligned}$$

and

$$\begin{aligned}
&\frac{1}{2} \big(\mathbf{u}_t^T (\mathbf{v} - \mathbf{v}_t) + (\mathbf{u} - \mathbf{u}_t)^T \mathbf{v}_t\big)^2 \\
&\le \|\mathbf{u}_t^T (\mathbf{v} - \mathbf{v}_t)\|_F^2 + \|(\mathbf{u} - \mathbf{u}_t)^T \mathbf{v}_t\|_F^2 \\
&\le \|\mathbf{u}_t\|_F^2 \|\mathbf{v} - \mathbf{v}_t\|_F^2 + \|\mathbf{v}_t\|_F^2 \|\mathbf{u} - \mathbf{u}_t\|_F^2,
\end{aligned}$$

we have

$$\begin{aligned}
h(\mathbf{u}, \mathbf{v}) \le &\big(\frac{\lambda_1}{2} + \frac{1}{2} |R_{i,j} - \mathbf{u}_t^T \mathbf{v}_t| + \|\mathbf{v}_t\|_F^2\big) \|\mathbf{u} - \mathbf{u}_t\|_F^2 \\
&+ \big(\frac{\lambda_2}{2} + \frac{1}{2} |R_{i,j} - \mathbf{u}_t^T \mathbf{v}_t| + \|\mathbf{u}_t\|_F^2\big) \|\mathbf{v} - \mathbf{v}_t\|_F^2.
\end{aligned}$$

Because

$$|(\mathbf{u} - \mathbf{u}_t)^T (\mathbf{v} - \mathbf{v}_t)| \le (\|\mathbf{u} - \mathbf{u}_t\|_F^2 + \|\mathbf{v} - \mathbf{v}_t\|_F^2)/2,$$

we have

$$\begin{aligned}
&|2(\mathbf{u} - \mathbf{u}_t)^T \mathbf{v}_t + 2(\mathbf{v} - \mathbf{v}_t)^T \mathbf{u}_t + (\mathbf{u} - \mathbf{u}_t)^T (\mathbf{v} - \mathbf{v}_t)| \\
&\le |2(\mathbf{u} - \mathbf{u}_t)^T \mathbf{v}_t| + |2(\mathbf{v} - \mathbf{v}_t)^T \mathbf{u}_t| + |(\mathbf{u} - \mathbf{u}_t)^T (\mathbf{v} - \mathbf{v}_t)| \\
&\le \|\mathbf{u} - \mathbf{u}_t\|_F^2 + \|\mathbf{v}_t\|_F^2 + \|\mathbf{v} - \mathbf{v}_t\|_F^2 \\
&\quad + \|\mathbf{u}_t\|_F^2 + \frac{1}{2} [\|\mathbf{u} - \mathbf{u}_t\|_F^2 + \|\mathbf{v} - \mathbf{v}_t\|_F^2] \\
&\le \|\mathbf{u}_t\|_F^2 + \|\mathbf{v}_t\|_F^2 + 3\delta^2,
\end{aligned}$$

Then we can prove

$$\begin{aligned}
o(\mathbf{u}_t, \mathbf{v}_t) \le |o(\mathbf{u}_t, \mathbf{v}_t)| \le &\frac{1}{4} \big(\|\mathbf{u}_t\|_F^2 + \|\mathbf{v}_t\|_F^2 + 3\delta^2\big) \\
&\times \big(\|\mathbf{u} - \mathbf{u}_t\|_F^2 + \|\mathbf{v} - \mathbf{v}_t\|_F^2\big).
\end{aligned}$$

Because $||\mathbf{u} - \mathbf{u}_t||_F^2 \le \delta^2$ and $||\mathbf{v} - \mathbf{v}_t||_F^2 \le \delta^2$, we can get

$$\begin{aligned}
\hat{f}_{i,j}(\mathbf{u}, \mathbf{v}) \le \ &\hat{f}_{i,j}(\mathbf{u}_t, \mathbf{v}_t) + \nabla_{\mathbf{u}}^T \hat{f}_{i,j}(\mathbf{u}_t, \mathbf{v}_t)(\mathbf{u} - \mathbf{u}_t) \\
&+ \nabla_{\mathbf{v}}^T \hat{f}_{i,j}(\mathbf{u}_t, \mathbf{v}_t)(\mathbf{v} - \mathbf{v}_t) \\
&+ (\frac{5}{4} \|\mathbf{v}_t\|_F^2 + \frac{1}{4} \|\mathbf{u}_t\|_F^2 + \frac{3}{4}\delta^2 + \frac{1}{2}\lambda_1 \\
&\quad + \frac{1}{2} |R_{i,j} - \mathbf{u}_t^T \mathbf{v}_t|) \|\mathbf{u} - \mathbf{u}_t\|_F^2 \\
&+ (\frac{5}{4} \|\mathbf{u}_t\|_F^2 + \frac{1}{4} \|\mathbf{v}_t\|_F^2 + \frac{3}{4}\delta^2 + \frac{1}{2}\lambda_2 \\
&\quad + \frac{1}{2} |R_{i,j} - \mathbf{u}_t^T \mathbf{v}_t|) \|\mathbf{v} - \mathbf{v}_t\|_F^2.
\end{aligned}$$

So if we let

$$\begin{aligned}
\frac{1}{\tau_t} \ge \max\{ &2m_i \big[\frac{5}{4} \|\mathbf{v}_t\|_F^2 + \frac{1}{4} \|\mathbf{u}_t\|_F^2 + \frac{3}{4}\delta^2 + \frac{1}{2}\lambda_1 \\
&+ \frac{1}{2} |R_{i,j} - \mathbf{u}_t^T \mathbf{v}_t|\big], \\
&2n_j \big[\frac{5}{4} \|\mathbf{u}_t\|_F^2 + \frac{1}{4} \|\mathbf{v}_t\|_F^2 + \frac{3}{4}\delta^2 + \frac{1}{2}\lambda_2 \\
&+ \frac{1}{2} |R_{i,j} - \mathbf{u}_t^T \mathbf{v}_t|\big]\},
\end{aligned}$$

we can prove that

$$G^p(\mathbf{U}, \mathbf{V}^p, \mathbf{\Theta}_t^p, \overline{\mathbf{V}}_t, \tau_t | \mathbf{U}_t, \mathbf{V}_t^p) \ge L^p(\mathbf{U}, \mathbf{V}^p, \mathbf{\Theta}_t^p, \overline{\mathbf{V}}_t).$$

That is to say, τ_t should be smaller than some value which is dependent on the current $\mathbf{U}_t$ and $\mathbf{V}_t^p$.

The second equation in Lemma 1 can be easily proved. □

B. PROOF OF LEMMA 2

PROOF. From Lemma 1, we have

$$\begin{aligned}
&G^p(\mathbf{U}_{t+1}, \mathbf{V}_{t+1}^p, \mathbf{\Theta}_t^p, \overline{\mathbf{V}}_t, \tau_t | \mathbf{U}_t, \mathbf{V}_t^p) \\
&\qquad \ge L^p(\mathbf{U}_{t+1}, \mathbf{V}_{t+1}^p, \mathbf{\Theta}_t^p, \overline{\mathbf{V}}_t), \\
&G^p(\mathbf{U}_t, \mathbf{V}_t^p, \mathbf{\Theta}_t^p, \overline{\mathbf{V}}_t, \tau_t | \mathbf{U}_t, \mathbf{V}_t^p) = L^p(\mathbf{U}_t, \mathbf{V}_t^p, \mathbf{\Theta}_t^p, \overline{\mathbf{V}}_t).
\end{aligned}$$

Furthermore, we have

$$\begin{aligned}
&G^p(\mathbf{U}_{t+1}, \mathbf{V}_{t+1}^p, \mathbf{\Theta}_t^p, \overline{\mathbf{V}}_t, \tau_t | \mathbf{U}_t, \mathbf{V}_t^p) \\
&\qquad \le G^p(\mathbf{U}_t, \mathbf{V}_t^p, \mathbf{\Theta}_t^p, \overline{\mathbf{V}}_t, \tau_t | \mathbf{U}_t, \mathbf{V}_t^p).
\end{aligned}$$

Hence, we can get

$$L^p(\mathbf{U}_{t+1}, \mathbf{V}_{t+1}^p, \mathbf{\Theta}_t^p, \overline{\mathbf{V}}_t) \le L^p(\mathbf{U}_t, \mathbf{V}_t^p, \mathbf{\Theta}_t^p, \overline{\mathbf{V}}_t).$$

□