Parallel and Distributed Collaborative Filtering: A Survey

EFTHALIA KARYDI and KONSTANTINOS MARGARITIS, University of Macedonia

Collaborative filtering is among the most preferred techniques when implementing recommender systems. Recently, great interest has turned toward parallel and distributed implementations of collaborative filtering algorithms. This work is a survey of parallel and distributed collaborative filtering implementations, aiming to not only provide a comprehensive presentation of the field's development but also offer future research directions by highlighting the issues that need to be developed further.

Categories and Subject Descriptors: A.1 [**Introductory and Survey**]; D.1.3 [**Programming Techniques**]: Concurrent Programming—*Distributed programming, Parallel programming*

General Terms: Algorithms, Documentation

Additional Key Words and Phrases: Collaborative filtering, recommender systems

ACM Reference Format:
Efthalia Karydi and Konstantinos Margaritis. 2016. Parallel and distributed collaborative filtering: A survey. ACM Comput. Surv. 49, 2, Article 37 (August 2016), 41 pages.
DOI: http://dx.doi.org/10.1145/2951952

1. INTRODUCTION

The quality of a recommender system's output highly depends on thequantity of used data. The more data available to a recommender system, the better the recommendation. Considering the need to address continuously growing amounts of data, the design of parallel and distributed recommender systems has become necessary. Parallel and distributed computing techniques can be combined with each other for the purpose of exploiting their advantages, and various modifications can be applied to existing algorithms to better fit the requirements of the utilized techniques. Furthermore, it is crucial for the development of high-quality recommender systems to utilize the available heterogeneous infrastructures. The study and design of parallel algorithms and implementations that will address emerging problems and exploit the advantages of new technologies is thus important.

Among the benefits that are expected to be obtained through the use of parallel and distributed systems in the field of recommender systems are the following:

—Faster result delivery is achieved, owing to more computational power and memory being available when using parallel and distributed systems.
—Greater amounts of data can be used—a fact that is expected to lead to greater efficiency.

Authors' address: E. Karydi and K. Margaritis, Department of Applied Informatics, Parallel and Distributed Processing Laboratory, University of Macedonia, 156 Egnatia Street, GR-54636, Thessaloniki, Greece; emails: karydithalia@gmail.com; kmarg@uom.gr.

DOI: http://dx.doi.org/10.1145/2951952

—The simultaneous execution of different algorithms can be facilitated, which can lead to the design of new algorithms that will utilize the results of all simultaneously running algorithms. Therefore, the use of different data sources will be easier, as will be the variety of item types that can be recommended.
—The development of new algorithms and improvements in existing ones will be accomplished through the study of the choice, design, and parallelization of algorithms suitable for the respective system.

Each parallel and distributed computing technique has advantages and disadvantages that must be considered when choosing the most appropriate technology or an adequate combination of different technologies to address each problem. The use of distributed systems is appropriate when employing algorithms that allow for data distribution or when different parts of the calculations can be performed independently of each other. However, the communication cost among nodes may be high and may even dominate the performance. Multithreading achieves short runtimes; however, special care must be given to avoid memory conflicts and race conditions. The use of frameworks for massively parallel processing augments the processing speed and facilitates large data handling, yet the algorithm must be adequate for implementation over the selected framework or appropriately modified. Graphic processing unit (GPU) usage can result in impressively fast processing, especially when an algorithm employs matrix-vector computations. Memory accesses must be controlled to achieve the best possible performance. The selection of the appropriate architecture to be used depends on the problem faced and on the algorithm chosen for parallelization. Parallel and distributed computing techniques must be carefully selected to help improve the overall performance.

1.1. Parallel and Distributed Computing

Currently, it is rare that a sequential computer can provide the processing power that many scientific and commercial applications need. A solution that addresses this constraint is using multiple interconnected processors to simultaneously execute tasks. This technique is called *parallel and distributed computing*. The distinction between parallel and distributed computing involves many overlaps and often is not clear [Kumar et al. 1994; Coulouris et al. 2011].

A distributed system is a collection of networked computers. When executing a task over a distributed system, communication and synchronization among the involved processors is performed by exchanging data messages over a common network interface.

The most common architectures used in distributed systems are client-server and peer-to-peer (P2P) architectures. In client-server architectures, a processor called the *server* provides a service and is used as a coordinator among the other processors, which are called *clients*. The clients communicate with the server and perform the tasks assigned to them. P2P systems differ from the client-server systems in that all of the system's processors coordinate to complete a task.

A computer cluster is a system of interconnected computer nodes that work as a single computing resource. According to the architecture of the computer nodes, a cluster can be homogeneous when all nodes have a similar architecture and heterogeneous when nodes of different architectures are used. Clusters are often referred to as distributed memory parallel systems. Communication and synchronization between computer nodes are handled by MPIs.

On shared memory systems, several processors share a common memory space. The communication between the processors is implicit, and synchronization is achieved by accessing different shared memory locations and using atomic operations and barriers.

Parallel applications are often accelerated through the use of one or more GPUs together with a central processing unit (CPU). GPUs are parallel co-processors that need to coordinate with a CPU and cannot be used without a CPU. GPUs follow a data parallel programming model. GPUs are based on scalable arrays of multithreaded streaming multiprocessors (SMs), which consist of several stream processors (SPs). Various libraries and interfaces exist for writing programs that can be executed across a system that consists of both CPUs and GPUs.

Many frameworks have been developed to facilitate the processing of high-volume data. In addition to the well-established message passing interfaces (MPIs), which are a standard in regard to communication among computer nodes, several frameworks have been developed, each one specialized for different programming models. Common technologies for shared memory parallel programming are OpenMP, Pthreads, and Java Threads, whereas CUDA and OpenCL are used for GPU computing. For distributed system development, Java-based tools seem to dominate.

One of the most popular frameworks is Hadoop [Apache Hadoop 2016]. Hadoop enables the distributed processing of large datasets across computer clusters. Hadoop is designed for the MapReduce model [Zhao and Pjesivac-Grbovic 2009], which represents data as key-value pairs and performs computations under map and reduce phases. Mahout [Apache Mahout 2016] is a framework adequate for the development of machine learning applications. Recently, many Mahout algorithms have been built on the Scala and Spark frameworks. The Spark framework [Apache Spark 2009] can run on Hadoop clusters or stand alone. Spark is adequate for both directed acyclic graph (DAG) and MapReduce applications. Another framework adequate for DAG-based applications is Storm [Apache Storm 2012]. A Storm cluster performs under the master-worker model and processes streaming data. Stratosphere [2009] is a large data analytics framework for distributed data processing. Stratosphere is adequate for both DAG and MapReduce implementations. GraphLab [Low et al. 2010] is a framework for graph-based implementations and for algorithms based on the bulk synchronous parallel (BSP) model. GraphLab is targeted at machine learning graph iterative algorithms for sparse data.

All of the preceding frameworks have been designed for creating scalable and fault-tolerant applications that are able to handle big data while minimizing the programmer's effort. Selecting between frameworks depends on the algorithms to be implemented and whether the framework is designed for graph, MapReduce, or BSP models. Another aspect that may influence a user's decision in selecting a framework can be the programming languages supported by each framework.

The continuous development of interfaces and frameworks for parallel and distributed programming offers a variety of options to programmers. Often, a combination of different technologies is the best solution for an application. Heterogeneous computing is the combination of different technologies using different types of processors. A heterogeneous application is developed following more than one parallel or distributed programming technique.

1.2. Recommender Systems

Recommender systems are mechanisms used to produce item recommendations to their users. Their purpose is to make personalized recommendations that will be interesting and simultaneously useful. This fact consists the difference between recommender systems and information retrieval search engines [Ricci et al. 2011].

Recommender systems are categorized into the following classes according to the techniques applied. *Collaborative filtering* recommender systems exploit the fact that users with similar past preferences are likely to have common preferences again. *Content-based* recommender systems calculate item similarities based on item features. *Demographic* recommender systems use the users' demographic information.

Knowledge-based recommender systems utilize specific domain knowledge that specifies to what extent an item is useful to the user. *Community-based* recommender systems provide recommendations based on the preferences of groups of users that have some common properties. All of preceding categories can be combined with each other, and a recommender system that belongs to more than one category is called a *hybrid* recommender system.

Collaborative filtering techniques are among the most popular techniques applied to recommender systems [Sarwar et al. 2000a]. Collaborative filtering recommender systems are further classified into *model-based* and *memory-based* systems. *Hybrid* collaborative filtering recommender systems, which combine model- and memory-based methods, have been developed. Su and Khoshgoftaar [2009] provide a classification of the collaborative filtering algorithms into the preceding categories.

Memory-based techniques are also mentioned as neighborhood-based methods. The dataset is used to calculate the similarity of the users with the active user. The active user is the user for whom the recommendation is produced. Then, a neighborhood is formed by the k users that are most similar to the active user. Finally, the predictions of the ratings that the active user would give to the items are computed. The similarity is more often measured by the Pearson correlation coefficient or by cosine vector similarity [Vozalis and Margaritis 2003], which are both variants of the inner product. The most popular algorithms that belong to this category are the item-based, user-based, and slope one algorithm. These algorithms can employ any of the similarity measures. The user- and item-based algorithms are often encountered in top-N approaches, where a set of N items is recommended.

Model-based techniques use the dataset to train a model and produce predictions according to the model. The objective of the model is to represent the user's behavior by recognizing the behavior patterns that occur in the training set, and the model benefits from the observed patterns to create predictions for the missing ratings. Various machine learning and data mining algorithms are used to create the model. Linear algebra models, such as singular value decomposition (SVD), principal component analysis (PCA), latent semantic analysis (LSA), and Latent Dirichlet allocation (LDA), are very often used to represent users and items by an f-dimensional latent factor space. To train these models, two commonly used methods are stochastic gradient descent (SGD) and alternating least squares (ALS). Models based on matrix factorization techniques are often preferred because they offer high accuracy and scalability [Ricci et al. 2011]. Other model-based techniques are Bayesian networks, clustering methods, and association rule-based methods [Su and Khoshgoftaar 2009].

Although the field of recommender systems has been significantly developed, certain problems remain unsolved. Great concern is given to aspects such as the quality of the recommendations, sparsity of the data, scalability, synonyms, and addressing new users and items, which are issues that have required attention since the beginning of recommender systems research [Sarwar et al. 2000a; Vozalis and Margaritis 2003].

Data sparsity is a growing problem that still needs to be addressed. Usually, the information that users provide to the recommender system is minimal considering the abundance of items that exist. This fact leads to very sparse data, which degrade the overall performance. Although many techniques have been developed to address data sparsity, it still remains a hot issue in the field of recommender systems.

Both the number of users and the number of items are continuously increasing. Thus, the need for fast and scalable computations is important. Currently, recommendations are expected to be produced extremely fast for a recommender system to be able to function properly online. Great effort must be given to develop efficient and scalable algorithms.

The variety of technologies that exist can provide great advantages. To exploit them in an efficient manner, the use of heterogeneous systems has increased. Thus, the algorithms should be redesigned to properly adjust to the needs that emerge from the use of heterogeneous systems.

Although research in the field of recommender systems has been conducted for more than 20 years, the issues that still offer room for improvement are not few in number. To address data abundance and to keep the time needed to produce the recommendations low, parallel and distributed systems are increasingly being used. In the following sections, approaches to recommender systems that employ parallel and/or distributed techniques will be surveyed to provide a concise view of the developments of the field and to highlight the areas that require further research.

1.2.1. Evaluation Metrics. How to evaluate recommender systems is an issue that attracts great interest. Recommender systems can have various properties, such as being trustful; recommending novel, useful, and interesting products; and being scalable. When designing a recommender system, one should decide which of the factors that characterize the recommender system are important for his implementation and therefore should select the adequate evaluation metrics to test whether the implementation meets the required criteria. A great variety of measures exists to evaluate each of the properties that a recommender system can have. The difficulty of applying a common evaluation framework for all recommender systems is revealed by considering the polymorphic nature that a recommender system can have and the variety of metrics.

One of the most important evaluation measurements is accuracy. Accuracy can measure how well a recommender system predicts a rating and is measured by means of mean absolute error (MAE) or root mean squared error (RMSE). Measures also exist that express how often a recommender system makes good or wrong recommendations. Metrics that classify accuracy are the F-measure, precision, recall, receiver operating characteristic (ROC curves) and area under the ROC curve (AUC) [Herlocker et al. 2004].

Since the fast delivery of results is very important, time is an aspect that is often measured. Usually the total elapsed time is measured, and the time of various tasks, such as the prediction delivery, the computation, and the communication time, is analyzed. Furthermore, when parallel and distributed programming techniques are used, the corresponding metrics, such as speedup and isoefficiency, are also employed.

1.2.2. Datasets. A great variety of datasets is used in recommender systems' research. Some of them contain demographic data or timestamps, whereas others emphasize in associations among the users. Additionally, the different order of magnitude and diversity on the rating scale, as long as the variety in sparsity and attributes provided in each dataset consist of reasons for which the use of more than one dataset to evaluate a recommender system, is fundamental.

The most commonly used datasets are the Netflix and MovieLens. The Netflix dataset was used for the Netflix Prize competition [Netflix 2009] and contains more than 480,000 users, 17,000 items, and 100 million ratings. Unfortunately, the Netflix dataset is no longer available. GroupLens Research [GroupLens 1997] has released the MovieLens datasets, which are offered in various sizes shown in Table I. The MovieLens 10M dataset has been recently extended to MovieLens 2k, which associates the movies of MovieLens dataset with their corresponding Web pages at the Internet Movie Database (IMDb) [IMDb 1990] and the Rotten Tomatoes movie review system [Rotten Tomatoes 1998].

General information about the most commonly used datasets in recommender systems can be seen in Table I.

Table I. Basic Information on Datasets

Dataset	Users	Items	Records	Scale	Value
Netflix	480,189	17,770	100,000,000	1–5	Integer
MovieLens 100k	943	1,682	100,000		
MovieLens 1M	6,040	3,900	1,000,000	1–5	Integer
MovieLens 10M	71,567	10,681	10,000,000		
MovieLens 20M	138,493	27,278	20,000,263	1–5	Integer
MovieLens 2k (HetRec 2011)	2,113	10,197	855,598	0–5	Real
Book-Crossing	278,858	271,379	1,149,780	1–10	Integer
Jester	73,496	100	4,100,000	(−10)–(+10)	Real
EachMovie	72,916	1,628	2,811,983	0–5	Integer
Yahoo! Music KDD Cup 2011					
Track 1	1,000,990	624,961	262,810,175	1–5	Integer
Track 2	249,012	296,111	61,944,406	1–5	Integer
Flixster	2,523,386	49000	8,200,000	1–5	Real
Delicious 2k (HetRec 2011)	1,867	69,226 URLs	—	—	—
Last.fm 2k (HetRec 2011)	1,892	17,632 artists	—	—	—
Million Song Dataset [McFee et al. 2012]	1,129,318	386,133	49,824,519	—	Integer

Table II. Classification of the Implementations

	Collaborative Filtering		
	Memory Based	*Model Based*	*Hybrid*
Distributed	13	4	7
Parallel			
Distributed memory		10	1
Shared memory	1	9	
GPU	6	10	
Platform Based	8	15	1
Heterogeneous	2	3	

1.3. Classification Scheme

The remainder of this article is organized as follows. Section 2 provides a brief collection of the survey approaches found in the literature that concern recommender systems. As can be noticed, none of these works addresses parallel and distributed collaborative filtering recommender systems. Section 3 presents the distributed implementations. Section 4 concerns the parallel implementations and separates them into three categories according to whether they are implemented in distributed memory environments or in shared memory environments and whether they utilize GPU accelerators. Platform-based approaches are discussed in Section 5, and Section 6 presents the heterogeneous approaches that belong to more than one of the preceding categories. In all sections, the implementations are classified according to which type of collaborative filtering technique belongs to the algorithm implemented. The structures according to which the implementations are classified can be found in Table II. In the

same table, one can also find the number of implementations that have been classified into each category. Finally, in Section 7, the conclusions of the survey are presented.

To the best of our knowledge, the present work represents the first attempt to collect the parallel and distributed implementations of collaborative filtering recommender systems. Studying existing implementations is expected to lead to the indication of further areas of study and to highlight the trends of recent research, as well as the gaps and the difficulties facing the field.

2. RELATED WORK

This section is devoted to briefly outlining the surveys concerning recommender systems. The surveys are classified into three categories. The first category includes surveys that provide a general introduction to recommender systems. The second category consists of surveys that address recommender systems of particular methods, and the third category lists surveys of applications of recommender systems.

2.1. General Introductory Surveys

One of the early surveys addressing recommender systems is the survey presented by Adomavicius and Tuzhilin [2005]. Recommender systems are classified into three categories: content-based, collaborative, and hybrid implementations. The constraints of each category are discussed, and possible ways of improving the recommendation methods are proposed. Park et al. [2012] present a literature review of works concerning recommender systems, classified according to their publication year, the data mining techniques that they applied, and the nature of the recommended items. Recent advances in the field are surveyed by Lü et al. [2012], whereas Bobadilla et al. [2013] provide an overview of the evolution of the recommender systems field, therein emphasizing the works that exploit social information and showing the importance of the various sources of information for the recommendation process.

A detailed presentation of the field of recommender systems and the most popular techniques used, such as collaborative filtering, content-based filtering, data mining, and context-aware systems, are detailed in Ricci et al. [2011] and Jannach et al. [2011]. Various applications are described, and a variety of topics are addressed. However, the scalability of the algorithms is not covered, and no chapter devoted to parallel and distributed applications in the field of recommender systems can be found in these works.

2.2. Surveys Concerning Methods of Recommender Systems

Recommender systems that combine different recommendation techniques are presented by Burke [2002]. A comparison of the different recommendation techniques is provided, and existing hybrid approaches are briefly presented. This survey proved that there were many combinations of techniques to be explored and outlined the needs of the field of hybrid recommender systems. Context-aware technology-enhanced recommender systems are discussed by Verbert et al. [2012], and a classification framework of the context information that divides the contextual information into eight categories is introduced. Tag-aware recommender systems are surveyed by Zhang et al. [2011], and Fernandez-Tobias et al. [2012] propose a taxonomy for cross-domain recommender systems and survey recent approaches. An overview of collaborative filtering and both content-based and hybrid recommender systems is given by Thorat et al. [2015].

A study on heterogeneous recommender systems is conducted by Bellogín et al. [2013a]. The effectiveness of various sources of information is investigated, and a variety of content-based, collaborative filtering, and social recommender systems are evaluated on different datasets. A comparative evaluation of social, collaborative filtering, and hybrid recommender systems is performed by Bellogín et al. [2013b]. Experimental

results are analytically presented and discussed in both works. Social recommender systems are also addressed by Bernardes et al. [2015], who provide a comparison of these systems using various datasets.

Collaborative filtering is studied by Su and Khoshgoftaar [2009]. The collaborative filtering techniques are classified into memory-based, model-based, and hybrid approaches, and the basic techniques of each category are described. A description and comparison of collaborative filtering algorithms are presented in many surveys [Cacheda et al. 2011; Candillier et al. 2007; Hameed et al. 2012]. The different algorithms used in user-based and item-based techniques are analyzed by Almazro et al. [2010], and the metrics used for evaluation are discussed.

A detailed and well-presented survey of collaborative filtering methods is provided by Ekstrand et al. [2011]. The different collaborative filtering methods are presented, and an analysis is given regarding which method is adequate for each problem that needs to be addressed. Detailed information is also provided on how to evaluate a recommender system.

Sachan and Richariya [2013] realized a survey on collaborative filtering approaches, mostly emphasizing how each approach addresses the most common challenges of collaborative filtering recommendations. This work concludes that more research is needed to address sparsity issues because sparsity affects the quality of the recommendations and also because datasets are expected to be even sparser in the future.

Shi et al. [2014] present the state of the art and discusses the challenges in the field of collaborative filtering recommendations for systems that in addition to using ratings also use information such as the features and characteristics of the users and items, or information related to the users' interactions.

Yang et al. [2014] survey collaborative filtering recommender systems that use information from social networks. The surveyed approaches are classified according to whether they use matrix factorization models or neighborhood-based approaches. Matrix factorization models are also presented by Bokde et al. [2015], who describe how they are used in collaborative filtering algorithms.

2.3. Surveys of Applications of Recommender Systems

Rao and Talwar [2008] analyzed the application domain of recommender systems. Almost 100 recommender systems are classified, and the majority belong to the Web recommendation, movie/TV recommendation, and information/document recommendation application domains. A survey of the work in the field of Web recommender systems is performed by Kumar and Thambidurai [2010], where a classification of Web recommender systems is outlined. Three techniques are mainly used: explicit and implicit profiling and legacy data. Lu et al. [2015] offers a survey of the current trends in recommender systems applications, where the applications are classified into eight categories, and the techniques that are used in each category are presented.

3. DISTRIBUTED IMPLEMENTATIONS

In this section, distributed implementations of collaborative filtering recommender systems are discussed. The implementations are classified into the collaborative filtering categories that are analyzed by Su and Khoshgoftaar [2009]. The implementations of each category are discussed according to their chronological appearance. This methodology is followed to demonstrate how the field of distributed recommender systems has evolved over the years and to offer an overview of what has been achieved.

Another factor that will be considered is the experiments that have been realized and the metrics that have been preferred for evaluation. Analyzing such factors will reveal the most-followed methods and will serve as a reference for conducting reproducible experiments. Table III provides a list of all implementations presented in this section.

Table III. List of Distributed Implementations

Reference	Category	Description
Olsson [1998]	HYBRID	Content-based, collaborative, and social filtering (item based)
Harth et al. [2001]	MODEL	iOwl tool, association rules
Tveit [2001]	MEMORY	User-based collaborative filtering (CF)
Canny [2002]	MODEL	P2P SVD
Han et al. [2004a], Han et al. [2004b]	MEMORY	User-based CF
Ali and van Stam [2004]	HYBRID	Item-based and Bayesian content-based filtering
Miller et al. [2004]	MEMORY	Item based
Berkovsky et al. [2005]	MEMORY	Traditional CF user based
Link et al. [2005]	HYBRID	Neighborhood and content based
Awerbuch et al. [2005]	HYBRID	Random product or user probation
Wang et al. [2006]	MEMORY	User-item relevance model and top-N CF (item based)
Castagnos and Boyer [2006]	HYBRID	Hierarchical clustering and user based
Berkovsky and Kuflik [2006]	MEMORY	Hierarchical formation in the CF algorithm (user based)
Xie et al. [2007]	MEMORY	CF with most same opinion and average rating normalization (user based)
Berkovsky et al. [2007a]	MEMORY	CF with data obfuscation (user based)
Berkovsky et al. [2007b]	MEMORY	CF with domain specialization (item based)
Castagnos and Boyer [2007]	MEMORY	User based
Ruffo and Schifanella [2009]	MEMORY	Affinity networks (user based)
Ahn and Amatriain [2010]	MEMORY	Expert CF (user based)
Liu et al. [2010]	HYBRID	Combination of user- and item based
Isaacman et al. [2011]	MODEL	Distributed gradient descent
Tomozei and Massoulié [2011]	MODEL	User profiling via spectral methods
Kumar et al. [2012]	HYBRID	Context-aware P2P service selection and recovery (CAPSSR)

Recommender systems developed using distributed computing techniques were initially proposed by Olsson [1998], Harth et al. [2001], and Tveit [2001]. In early distributed collaborative filtering approaches, no preference is given to any specific algorithmic category.

Olsson [1998] proposes a method that combines content-based collaborative filtering and social filtering. Harth et al. [2001] introduces a model-based recommender system named iOwl, which works both as a server and as a client and suggests links to Web pages to its users using association rules. These two approaches propose models that collect data from Web sites. As a result, the repetition of any conducted experiments will be difficult. A memory-based approach that uses the Pearson correlation coefficient on a P2P architecture similar to Gnutella [Ripeanu et al. 2002] is described by Tveit [2001]. The preceding approaches emphasize the description and analysis of the proposed model without conducting any experiments. Therefore, no evaluation is provided. However, those methods consist the beginning of the field of distributed recommender systems.

3.1. Distributed Memory–Based Collaborative Filtering

In this section, the distributed implementations of memory-based collaborative filtering algorithms are presented. Initially, traditional user-based and item-based collaborative filtering methods were chosen for implementation.

Implement the user-based algorithm on a P2P architecture through a distributed hash table (DHT) method [Han et al. 2004a; Han et al. 2004b]. Different parts of the user database are distributed to peers in such a way that all users in the same peer group have rated at least one item with the same value. After the similar peers are found, a local training set is constructed, and the similar users' vote vectors are retrieved and used to compute the prediction. Miller et al. [2004] use five P2P architectures to examine the item-based algorithm's performance. A model is created for the users while they are online. This model is used even if the users are offline. Berkovsky et al. [2005] apply the traditional collaborative filtering algorithm over a set of distributed data repositories. Data are distributed both geographically and by topic.

Although Han et al. [2004a] and Han et al. [2004b] use a different dataset than Miller et al. [2004] and Berkovsky et al. [2005], all implementations are evaluated using the MAE metric. Miller et al. [2004] also measure recall, coverage, and memory usage. It would be interesting to test all of the proposed algorithms on the same datasets to compare the prediction accuracy of the different approaches.

Next, more sophisticated ideas that combine traditional collaborative filtering algorithms with other methods have been developed. Wang et al. [2006] calculate item similarity using log-based user profiles collected from the Audioscrobbler community [Audioscrobbler 2002]. The items are distributed over a P2P network, and the relevance between two items is updated only when an item is downloaded by a peer. The similarities between items are stored locally in item-based tables. Finally, the top-N ranked items are recommended to the user. Berkovsky and Kuflik [2006] form a hierarchical neighborhood, which consists of super-peers and peer-groups. Super-peers are responsible for computations within their peer-group and aggregate their results before providing them to the active user. Xie et al. [2007] propose a distributed collaborative filtering algorithm based on traditional memory-based collaborative filtering. The proposed algorithm locates similar users using a DHT scheme. The number of users that contribute to the recommendation is reduced using the concept of the most same opinion. Thus, only the ratings of the users with the highest consistency with the active user are used. Furthermore, to avoid losing users who have similar taste but do not rate the items identically, average rating normalization is applied. Berkovsky et al. [2007a] combine the distributed storage of user profiles with data alteration techniques to mitigate privacy issues. This approach focuses on the effect of obfuscating the ratings on the accuracy of the predictions. Domain specialization over the items is developed by Berkovsky et al. [2007b] to confront the data sparsity problem. The rating matrix is partitioned into smaller matrices that contain ratings given to items that belong to a certain type. Wang et al. [2006] give the coverage and precision of the recommendations, and Berkovsky and Kuflik [2006], Berkovsky et al. [2007a], Berkovsky et al. [2007b], and Xie et al. [2007] use the MAE metric. Information on the variety of the datasets, the technologies, and the applied metrics can be found in Table IV.

A variation on the user-based collaborative filtering algorithm is proposed by Castagnos and Boyer [2007]. Each user has his own profile and a single ID. The users can affect the degree of personalization implicitly. The Pearson correlation coefficient is used for the similarity computation, and the nearest neighbors of the active user are selected. Four lists of IDs are kept for each user, representing the most similar users, users who exceed the minimum correlation threshold, the black-listed users, and users who have added the active user to their group profile. Because there is no need to store any neighbors' ratings or similarities, this model has the advantage of low memory requirements. The algorithm is evaluated on the MovieLens dataset, therein measuring the MAE metric and the computation time.

Table IV. Distributed Memory–Based Implementations

Algorithm	Technologies	Datasets	Metrics
User-based collaborative filtering (CF) [Tveit 2001]	Java	N/A	N/A
PipeCF [Han et al. 2004a] [Han et al. 2004b]	DHT	EachMovie	MAE
PocketLens Item based [Miller et al. 2004]	Chord architecture for P2P file-sharing networks	MovieLens	Neighborhood similarity MAE, recall, coverage Memory usage, prediction time
Traditional CF [Berkovsky et al. 2005]	Loud voice platform	MovieLens	MAE
User-Item relevance model [Wang et al. 2006]	N/A	Audioscrobbler	Coverage Precision
Distributed hierarchical neighborhood formation in the CF algorithm [Berkovsky and Kuflik 2006]	Java simulation	MovieLens EachMovie Jester	MAE
DCFLA [Xie et al. 2007]	Algorithmic simulation	EachMovie	MAE
Distributed storage of user profiles [Berkovsky et al. 2007a]	Java simulation	MovieLens	MAE
Item clustering [Berkovsky et al. 2007b]	Java simulation	EachMovie	MAE
User-based AURA [Castagnos and Boyer 2007]	JXTA Platform	MovieLens	MAE Computation time
Affinity networks [Ruffo and Schifanella 2009]	Modification of Phex (Java file-sharing app)	Self-collected	Average accuracy
Expert CF [Ahn and Amatriain 2010]	RIA (Java, RESTful, XML-RPC)	Collected from metacritic.com, rottentomatoes.com	N/A

Ruffo and Schifanella [2009] describe a P2P recommender system that instead of employing user profiles to produce the recommendations uses affinity networks between the users. The affinity networks are generated according to the files that the peers are sharing. Ahn and Amatriain [2010] present a distributed expert collaborative filtering [Amatriain et al. 2009] recommender system. In expert collaborative filtering, the peer user ratings are replaced with ratings provided by domain experts. In this implementation, the expert ratings are acquired from Metacritic [1993]. The expert ratings are stored to the server in a matrix that is used by the clients during the recommendation process. The distributed expert collaborative filtering approach has the advantage that it deals with privacy issues well because user profile information is maintained on the users' machines.

3.2. Distributed Model–Based Collaborative Filtering

This section briefly presents the distributed model–based collaborative filtering implementations. Canny [2002] introduces the first distributed recommender system

Table V. Distributed Model–Based Implementations

Algorithm	Technologies	Datasets	Metrics
Association rules [Harth et al. 2001]	Python, iOwl	N/A	N/A
P2P SVD [Canny 2002]	Matlab	EachMovie	MAE Average recommendation time
Distributed gradient descent [Isaacman et al. 2011]	Facebook app WebDose	Netflix	RMSE Probability distribution Estimation of rating
Similarity-based profiling [Tomozei and Massoulié 2011]	Mathematical simulation	Netflix (synthetic)	Convergence of the asynchronous distributed algorithm

implementation, wherein a P2P SVD model is proposed. This work focuses on privacy issues, and the recommendations are provided from a distributed computation of an aggregate model of user preferences.

Among the most popular matrix factorization techniques is the SGD algorithm. A distributed implementation of this algorithm is proposed by Isaacman et al. [2011], where the information that users provide over items is only available to the users who produced these items.

Another dimensionality reduction algorithm was developed by Tomozei and Massoulié [2011]. A distributed user profiling algorithm creates a profile vector for each user that represents her taste. Considering a network described by an undirected graph, a similarity value is calculated between all nodes that are connected. The eigenvectors of the adjacency matrix defined from the similarity values are computed in a distributed manner and are used to form the recommendations.

The datasets and metrics used in the preceding implementations can be found in Table V.

3.3. Hybrid Distributed Collaborative Filtering Methods

In addition to Olsson [1998], a few more hybrid distributed methods have been developed. These implementations can be found in Table VI.

Ali and van Stam [2004] follow a client-server architecture, where item correlations are computed at the server side and are used on the client side to make the predictions. No evaluation of the model is provided.

Link et al. [2005] combine memory-based collaborative filtering using neighbors with content-based collaborative filtering. The "mailing list" model and the "word-of-mouth" model are described. Users share information with their neighbors according to one of the two models. The intention of the distributed recommender systems that described in this work is to provide item information to as many users as possible, who are expected to have an interest in the items. Unfortunately, no details are given on the implementation, and its performance needs to be evaluated.

Awerbuch et al. [2005] describe a P2P distributed algorithm that focuses on the minimization of the recommendation complexity by avoiding the evaluations provided by the untrusted users. However, the algorithm is only described theoretically and is not implemented.

User-based collaborative filtering employing the Pearson correlation coefficient is combined with a hierarchical clustering algorithm by Castagnos and Boyer [2006]. The user profiles are sent to the server, and the system creates virtual communities using the hierarchical clustering algorithm. The classification of the active user to a group

Table VI. Distributed Hybrid Implementations

Algorithm	Technologies	Datasets	Metrics
Content-based filtering Collaborative filtering and social filtering [Olsson 1998]	Agent based	N/A	N/A
Item-based Bayesian content-based filtering [Ali and van Stam 2004]	Proprietary	TiVo data	N/A
User neighborhood and content-based filtering [Link et al. 2005]	Mathematical simulation	N/A	N/A
User-based hierarchical clustering [Castagnos and Boyer 2006]	Java	MovieLens	MAE Computation time
Random product or user probation [Awerbuch et al. 2005]	Mathematical simulation	N/A	N/A
User- and item-based combination [Liu et al. 2010]	N/A	MovieLens	MAE
Context-aware P2P service CAPSSR [Kumar et al. 2012]	N/A	MovieLens Jester	Scalability Accuracy DFM, precision Mean waiting time

occurs on the client side. The predictions are made according to the distances between the active user and the closest group's users.

Liu et al. [2010] combine user-based and item-attribute–based ratings by using the item similarities to predict a user's rating for an item. The users are clustered according to user similarity values, therein using the k-means algorithm and demographic data. The described approach focuses on the user cluster formation, and no details are given on the P2P communication protocol.

Kumar et al. [2012] propose an algorithm for context-aware P2P service selection (CAPSSR). Users can access various services available on the Internet. After using one service, the service's rating is increased or decreased depending on whether the use of the service was successful. For the evaluation of the algorithm, the MovieLens and Jester datasets are used. Scalability, accuracy, efficiency, and mean waiting time are evaluated. This is the only distributed implementation that provides information about the algorithm's scalability.

The majority of distributed memory–based approaches are developed using the Java language, whereas there seems to be no language preference in memory-based and hybrid distributed implementations. The distributed approaches are mainly evaluated by accuracy metrics, with the MAE measured in particular. Few approaches evaluate the accuracy using the RMSE or other metrics, and the most popular datasets are the EachMovie and MovieLens datasets. Scalability has only recently been considered.

Table III lists all distributed collaborative filtering implementations discussed in Section 3 in chronological order. An initial preference for memory-based techniques is observed. However, in recent years, interest seems to have turned to model-based and hybrid approaches. This most likely occurs because dimensionality reduction techniques are more suitable for addressing the ever-increasing amount of data to be processed. Thus, the model-based approaches seem to be more promising for delivering results faster than memory-based approaches.

Table VII. List of Implementations on Distributed Memory Systems

Reference	Category	Description
George and Merugu [2005]	MODEL	Bregman co-clustering
Zhou et al. [2008]	MODEL	ALS-WR
Chen et al. [2008]	HYBRID	Combinational collaborative filtering
Kwon and Cho [2010]	MODEL	Bregman co-clustering
Liu et al. [2011]	MODEL	PLDA+
Yu et al. [2012]	MODEL	Coordinate descent CCD++
Teflioudi et al. [2012]	MODEL	DALS, ASGD, DSGD++
Narang et al. [2012]	MODEL	Co-clustering
Ho et al. [2013]	MODEL	SGD, LDA, CGD
Yun et al. [2014]	MODEL	RobiRank p
Yu et al. [2014]	MODEL	DS-ADMM

Among the memory-based algorithms, traditional user- and item-based algorithms are deployed more often than the top-N approaches. The majority of distributed memory–based collaborative filtering approaches employ the MAE metric to measure the recommendation accuracy. Other metrics, such as recall, coverage, and precision, are being used less often. However, none of the experiments include speedup analysis, and computation and communication time are rarely considered. Emphasis is given to privacy issues by distributing parts of the users' information to the available peers. Occasionally, the P2P architecture is simulated by multithreaded applications, although no preference to any specific technology is shown. The MovieLens and EachMovie datasets are preferred for the larger portions of the experiments.

The model-based algorithms developed for distributed systems are not enough to offer sufficient conclusions. However, it is noticeable that none of the implementations employs clustering techniques and that the dimensionality reduction techniques seem to attract greater interest. In addition to the accuracy metrics, factors such as the time needed for a recommendation and the algorithm's convergence are also measured. Yet no preference is found for any specific metrics.

Table VI presents the hybrid distributed collaborating filtering approaches. As can be seen, information on the technologies and datasets used is incomplete, and no common framework exists for performance evaluations. Some of the proposed methods are mathematical simulations and are not implemented. In addition, the small number of hybrid distributed approaches reveals a gap that needs to be filled. Investigating the performance of other hybrid implementations could prove to be useful.

4. PARALLEL IMPLEMENTATIONS

4.1. Distributed Memory Implementations

This section presents the parallel implementations that are built on distributed memory systems. A list of these approaches is provided in Table VII, and additional information can be found in Table VIII. As shown in these tables, no memory-based algorithms are implemented in distributed memory systems, and a clear preference is noticed for model-based algorithms. Here, the implementations are presented according to the implemented algorithm.

Clustering is a model-based collaborative filtering method that is used quite often. The Bregman co-clustering algorithm [Banerjee et al. 2004] is parallelized by George and Merugu [2005] and Kwon and Cho [2010]. George and Merugu [2005] creates the user and item neighborhoods simultaneously by dividing the submatrices of the rows and columns of the ratings matrix among the processors. A comparison of the proposed algorithm with SVD [Sarwar et al. 2000b], NNMF [Hofmann 2004], and

Table VIII. Parallel Implementations on Distributed Memory Environments

Algorithm	Technologies	Datasets	Metrics
Parallel co-clustering Bregman [George and Merugu 2005]	C++, MPI LAPACK library	MovieLens Bookcrossing	MAE, average prediction time Training time Comparison to SVD, NNMF and classic correlation-based filtering
ALS-WR [Zhou et al. 2008]	Parallel Matlab, MPI	Netflix	RMSE
Combinational collaborative filtering (CCF) [Chen et al. 2008]	MPI	Orkut (synthetic)	Speedup, computation/ communication time analysis
Bregman co-clustering [Kwon and Cho 2010]	MPI	Netflix	Speedup Time per iteration
PLDA+ [Liu et al. 2011]	MPI	NIPS Wiki 20T Wiki 200T	Speedup, communication time, sampling time
Coordinate descent CCD++ [Yu et al. 2012]	C++ and MPI	MovieLens Netflix Yahoo! Music	Speedup, training time
DALS, ASGD, DSGD++ [Teflioudi et al. 2012]	C++ MPI	Netflix KDD Cup 2011 (Track 1)	Time per iteration, number of iterations, total time to converge
Co-clustering [Narang et al. 2012]	MPI	Netflix Yahoo KDD Cup	RMSE Speedup
SGD, LDA, CGD [Ho et al. 2013]	N/A	Netflix	Computation time vs. network waiting time
RobiRank p [Yun et al. 2014]	N/A	Million Song Dataset	Speedup, precision
DS-ADMM [Yu et al. 2014]	MPI	Netflix Yahoo! Music R1, R2	Speedup RMSE

classic correlation-based filtering [Resnick et al. 1994] is provided. Kwon and Cho [2010] perform the row and column cluster assignments in parallel by also dividing the rows and columns among processors. In both implementations, MPI is used.

Another co-clustering–based collaborative filtering algorithm is proposed and examined by Narang et al. [2012]. The algorithm's performance is compared to the authors' previous work [Narang et al. 2011b]. The initial ratings matrix is partitioned according to a certain number of rows and columns, and the algorithm described is applied to each partition by Narang et al. [2011b]. The row and column clusters formed in each partition are merged with the neighboring partition. This procedure is followed by various levels of row and column clusters until the complete matrix is obtained as a single partition. Then, the flat parallel co-clustering is applied once more. This hierarchical co-clustering algorithm attempts to reduce the communication and computation costs. The performance of the proposed algorithm is examined on the Netflix and Yahoo KDD Cup datasets. The experiments are conducted on the Blue Gene/P architecture, and RMSE is the accuracy metric used. Detailed scalability analysis is also provided.

A distributed LDA algorithm is described by Liu et al. [2011] and is implemented using MPI. This implementation improves the scalability of the author's previous effort [Wang et al. 2009] and reduces the communication time by applying methods such as data placement, pipeline processing, word bundling, and priority-based scheduling.

Zhou et al. [2008] implement the alternating least squares with weighted λ regularization algorithm (ALS-WR) using parallel Matlab. The updates of the U and M

matrices are parallelized, and the rows and columns of the ratings matrix are distributed over the cores.

The ALS and SGD algorithms used for matrix factorization are parallelized by Teflioudi et al. [2012]. The parallel ALS (PALS), parallel SGD (PSGD), distributed ALS (DALS), asynchronous SGD (ASGD), and DSGD-MR, along with its extension DSGD++, are described, implemented, and compared. All of the preceding algorithms are implemented in C++, and MPI is used for communication over the nodes of the distributed algorithms. The Netflix dataset and the dataset of Track 1 of the KDD Cup 2011 contest are used. The time required to complete an iteration, the number of iterations required to converge, and the total time to converge of the algorithms are compared.

Yu et al. [2012] propose a coordinate descent algorithm, CCD++, that approximates the ratings matrix by WH^T, therein updating one variable at a time while maintaining the other variables fixed. The algorithm is parallelized on an MPI cluster. Each machine updates different subvectors of the row vectors of W and H and broadcasts the results. The CCD++, ALS, and SGD algorithms are parallelized and compared. The training time and the speedup are measured. The MovieLens 10M, Netflix, and Yahoo! Music datasets are used for the experiments.

Yun et al. [2014] present a parallel version of an algorithm that predicts the songs a user is going to listen to. The proposed algorithm, RobiRank p, is a stochastic optimization algorithm for item ranking based on SGD. The data are distributed across p processors. Each processor runs SGD during a predefined time period, and at the end of this period, a synchronization of all processors takes place, and each processor independently updates a parameter ξ, which is used to obtain an upper bound of the objective function.

A stochastic matrix factorization model called the *distributed stochastic alternating direction methods of multipliers* (DS-ADMM) is proposed by Yu et al. [2014]. First, the data are distributed over the processors by dividing the ratings matrix and the matrix that contains the users' latent factors into submatrices. Each submatrix will contain a part of both matrices. Each node will have one of the submatrices and a local copy of the item latent factor matrix. Then, in each iteration, each node updates its corresponding user and item latent factor matrices, and the global item latent matrix is updated when all nodes have finished updating their local copies.

Among other machine learning algorithms, SGD, LDA, and CGD are also implemented using a parallel parameter server by Ho et al. [2013]. The algorithms are developed under a stale synchronous parallel (SSP) model.

Chen et al. [2008] propose a collaborative filtering method for community recommendation for social networking sites. Parallel Gibbs sampling and the parallel expectation maximization algorithm are combined. Experiments are performed on the Orkut dataset to measure the implementation's speedup. Furthermore, an analysis of the computation and communication time is provided. However, no information on the technologies used to achieve the algorithm's parallelization is given.

4.2. Shared Memory Implementations

Recommendation algorithms that have been implemented on shared memory architectures will be discussed in this section. A list of these implementations is given in Table IX.

Narang et al. [2010] present a parallel model-based collaborative filtering algorithm based on the concept decomposition technique for matrix approximation. This technique performs clustering with the k-means algorithm and then solves a least squares problem. The proposed algorithm consists of four multithreaded stages, concluding

Table IX. List of Implementations on Shared Memory Systems

Reference	Category	Description
Narang et al. [2010]	MODEL	Concept decomposition
Louppe and Geurts [2010]	MODEL	Asynchronous gradient descent
Recht and Re [2011]	MODEL	SGD
Recht et al. [2011]	MODEL	SGD
Karydi and Margaritis [2012a]	MEMORY	Slope one
Yu et al. [2012]	MODEL	Coordinate descent CCD++
Zhuang et al. [2013]	MODEL	FPSGD
Shi et al. [2013]	MODEL	GAPfm
Zhang et al. [2013]	MODEL	LMF
Lee et al. [2014]	MODEL	LCR

with the prediction phase. Pthreads is used to implement the proposed method, which is evaluated on the Netflix dataset. Training and prediction time are measured using the RMSE metric. A detailed scalability analysis is also presented.

Parallel gradient descent in a shared memory environment is applied by Louppe and Geurts [2010]. In this approach, if the parameter θ is already processed, the other processors skip the update, and the processor with the most queued updates is the next processor that obtains access to update θ. This method attempts to reduce the idle time of the processors.

Recht and Re [2011] implements incremental SGD on multicore processors. One core is assigned for the ordering and partitioning of the data into chunks. Nonoverlapping chunks are grouped into rounds, and each round's chunks are accessed by a different process.

Recht et al. [2011] implement SGD without locking access to shared memory. Although memory overwrites are not avoided, they are rare because of the data sparseness. Therefore, they do not cause errors in the computations.

Karydi and Margaritis [2012a] describe a multithreaded application of the memory-based slope one algorithm, implemented with the OpenMP library. Each thread assumes the computations on a different part of the ratings matrix. The MovieLens dataset is used for the performance and scalability evaluation, and the metrics used for the evaluation can be seen in Table X.

The CCD++ algorithm [Yu et al. 2012] described in Section 3.2 is also parallelized on a multicore system using the OpenMP library. Each core updates different subvectors of the row vectors of W and H. Parallel implementations of the CCD++, ALS, and SGD algorithms are compared based on the running time against RMSE and speedup. The datasets used for the experiments can be found in Table X.

A new parallel matrix factorization approach based on SGD is analyzed by Zhuang et al. [2013]. The FPSGD method is designed for shared memory systems and embodies two techniques: lock-free scheduling to avoid data imbalances and a partial random method to address memory discontinuity. A comparison of other parallel SGD methods [Gemulla et al. 2011; Recht et al. 2011; Yu et al. 2012] is provided, and after applying optimizations such as cache-miss reduction and load balancing, FPSGD is found to be more efficient. Information is given concerning the algorithm's runtime, and the RMSE is used to evaluate the implementation. The MovieLens, Netflix, and Yahoo! Music datasets are used for the experiments.

Zhang et al. [2013] present localized matrix factorization (LMF), a method that factorizes the matrices in block diagonal form before proceeding to a prediction. First, a large sparse matrix is transformed into a block diagonal form and then the diagonal blocks can be trained in parallel. Computational time and speedup are determined using eight threads.

Table X. Implementations on Shared Memory Systems

Algorithm	Technologies	Datasets	Metrics
Concept decomposition [Narang et al. 2010]	Pthreads	Netflix	RMSE, ccalability Prediction/training time
Asynchronous gradient descent [Louppe and Geurts 2010]	N/A	Netflix	Speedup, parallel efficiency RMSE, wall-clock time
Parallel SGD JELLYFISH [Recht and Re 2011]	N/A	MovieLens Netflix	Total CPU time RMSE
Multicore SGD Hogwild [Recht et al. 2011]	C++ Pthreads	Reuters RCV1 Netflix KDD Cup 2011 (Task 2) Jumbo (synthetic) Abdomen	Speedup
Slope one [Karydi and Margaritis 2012a]	OpenMP	MovieLens	Scalability, speedup Total elapsed time Prediction per second Prediction time per rating
Coordinate descent CCD++ [Yu et al. 2012]	C++ and OpenMP	MovieLens Netflix Yahoo! Music	Running time vs. RMSE, speedup
FPSGD [Zhuang et al. 2013]	C++ SSE instructions	MovieLens Netflix Yahoo! Music	Total time RMSE
GAPfm [Shi et al. 2013]	Matlab	Netflix MovieLens	Average iteration time
LMF	N/A	MovieLens Yahoo! Music DianPing	RMSE Speedup Computational time
LCR [Lee et al. 2014]	N/A	MovieLens EachMovie Yelp	Zero-one error Average precision DCG@k

Shi et al. [2013] propose the GAPfm algorithm, which uses latent factors to provide top-N recommendations. Parallel computing is used to update the user latent factors that the algorithm needs to learn simultaneously. The proposed algorithm is based on the GAP algorithm presented by Robertson et al. [2010], and its performance is compared to that of PopRec, SVD++, and CofiRank. The proposed algorithm is executed using eight cores, and its design for distributed systems lies within the authors' future plans.

Lee et al. [2013] propose a matrix factorization model assuming that the rating matrix is locally low rank and represented as a weighted sum of low-rank matrices. The proposed model, called *LLORMA*, is based on SVD. Lee et al. [2014] extend LLORMA and propose a method called *local collaborative ranking* (LCR). This method uses synchronization at the beginning of each iteration, and each local model is updated in parallel. The description of the method is not focused on the parallelization settings and the used technologies. It is interesting that the proposed method is evaluated using the Yelp dataset, which enables experiments under extreme sparsity levels. In addition, its accuracy and precision are evaluated using metrics that have not been used before in other approaches on shared memory systems. However, the algorithm's performance on parallel systems is not evaluated.

Table XI. List of Implementations on GPUs

Reference	Category	Description
Vinay et al. [2006]	MODEL	SVD
Lahabar and Narayanan [2009]	MODEL	SVD
Kato and Hosino [2010b]	MODEL	SVD
Kato and Hosino [2010a]	MEMORY	K-nearest neighbor
Hansen et al. [2011]	MODEL	Co-clustering
Li et al. [2011]	MEMORY	Top-N user-based random walk
Tripathy et al. [2012]	MEMORY	Item-based collaborative filtering (CF), user-based CF
Chua [2012]	MODEL	Approximate SVD
Cai et al. [2012]	MODEL	RBM-CF
Zastrau and Edelkamp [2012]	MODEL	SGD
Zhanchun and Yuying [2012]	MEMORY	User-based CF
Foster et al. [2012]	MODEL	Approximate SVD
Cai et al. [2013]	MODEL	RBM-CF
Nadungodage et al. [2013]	MEMORY	Item based
Noel and Osindero [2014]	MODEL	Dogwild (SGD)
Wang et al. [2014]	MEMORY	User based

4.3. GPU-Based Implementations

Recently, general-purpose computations on GPU devices have emerged as an attractive solution for parallel computing. The performance of implementations belonging to various areas of computer science has been significantly increased when GPUs are used. This section presents implementations of collaborative filtering algorithms that have been parallelized on GPU devices. First, the memory-based implementations will be described according to their chronological appearance and then the model-based approaches will be discussed according to the algorithm they implement. Table XI presents a list of all implementations on GPUs that are discussed next.

4.3.1. Memory-Based Implementations on GPU. The k-nearest neighbor problem is confronted by Kato and Hosino [2010a], where an algorithm that finds the k most similar users using GPUs is introduced. The Hellinger distance is employed, and the algorithm is implemented in CUDA. The problem of computing the distances is divided into grids. Each GPU processes a grid. Each grid is divided row-wise into blocks, which are assigned to thread blocks. Each thread assumes a row of the block. For the selection of the nearest neighbors, the threads in a block simultaneously process their corresponding parts of the data and realize the necessary computations.

Li et al. [2011] describe a hybrid parallel top-N recommendation algorithm that attempts to address the cold-start user problem and the scalability problem. The proposed algorithm combines user-based collaborative filtering with random walk on trust network and merges the results to provide the top-N recommended items. First, the user-based algorithm, where the similarities between users are computed based on the Pearson correlation, is run. A heap structure is used to assist in selecting a subset of similar users. Finally, random walks are used to define a subset of trusted users. The results obtained by the two algorithms are merged to provide the final top-N recommendations. All three parts of the algorithm are implemented in CUDA.

The traditional item- and user-based collaborative filtering algorithms are parallelized by Tripathy et al. [2012]. The performance of the proposed algorithms is examined using Intel's Single-Chip Cloud Computer (SCC) and using an NVIDIA Cuda-enabled GPGPU co-processor. The similarity measure used is the Pearson correlation coefficient. The identification of common items is usually achieved via brute force

Table XII. Memory-Based Implementations on the GPU

Algorithm	Technologies	Datasets	Metrics
K-nearest neighbor [Kato and Hosino 2010a]	CUDA	N/A	Total elapsed time
Top-N User-based collaborative filtering Random walk [Li et al. 2011]	C++, CUDA	Flixster	Recall Speedup
User based Item based [Tripathy et al. 2012]	CUDA	Flixter (synthetic) Bookcrossing (subset) MovieLens (subset)	Execution time Power/energy consumption Speedup
User based [Zhanchun and Yuying 2012]	CUDA	GroupLens (subset)	RMSE, execution time CPU/GPU time usage
Item based [Nadungodage et al. 2013]	CUDA C++	Flixster	Speedup Runtime
User based [Wang et al. 2014]	CUDA	MovieLens	Speedup Runtime

methods. This approach avoids such methods using an intermediate matrix. The number of co-rated items is calculated, and subsequently the intermediate matrix is used to calculate the correlation coefficient.

Another implementation of the user-based collaborative filtering algorithm on the GPU is approached by Zhanchun and Yuying [2012]. Three different approaches are investigated. First, the Pearson correlation coefficient is used. Then, implied similarities are calculated. Implied similarity is based on the common neighbors among users. Finally, the empty cells of the ratings matrix are filled with the value of the average rating for each user. The accuracy of the three approaches, as well as the total execution time on both CPUs and GPUs, are examined using a part of a dataset provided by GroupLens.

Nadungodage et al. [2013] develop two item-based GPU algorithms using a different compression technique for each algorithm. The first algorithm is implemented using bit packing, and the second algorithm uses a compact format to store the nonzero values. Three kernels are used in both versions. The first kernel calculates the weighted co-occurrence frequency of pairs of items, the second calculates the similarity values, and the third finds the top-k similarities. Only the first kernel exhibits differences in the two implemented algorithms. In the first algorithm, the co-occurrence of a pair of items is computed in each row in each thread block. Each row of threads computes the weighted sum in parallel. In the second algorithm, in each thread block, each row of threads processes the data that correspond to a user and updates the co-occurrences using atomic operations.

The user-based algorithm on GPUs is also developed by Wang et al. [2014]. Two kernels are used. The first kernel calculates the average rating of the users, and the second kernel computes the similarities. Each thread block calculates the partial Pearson correlation and stores it in shared memory. The partial results are accumulated until all data are processed.

Table XII shows the datasets on which the preceding implementations are used to conduct the experiments and the evaluation metrics that are used.

4.3.2. Model-Based Implementations on GPUs. Model-based collaborative filtering implementations on GPUs commenced with an approach to the SVD algorithm [Vinay et al. 2006]. First, a bidiagonalization of the ratings matrix is performed and then the bidiagonal matrix is diagonalized by an implicit-shifted QR algorithm. The diagonalization is

performed on a CPU. The time needed for the bidiagonalization according to the size of the matrix is measured. Information on how the parallelization on the GPU is achieved is not specified, nor is any information on the used dataset given.

Among the first implementations of SVD on the GPU is that described by Lahabar and Narayanan [2009]. The CUDA architecture and CUBLAS library are used. All data necessary to perform the bidiagonalization are stored in the GPU memory to avoid data transfer between CPUs and GPUs. The diagonalization of the bidiagonal matrix is also performed on the GPU. The rows of the matrix are divided into blocks, and each element of the block is processed by a different thread. The performance is compared to that of an optimized CPU implementation on Matlab and to Intel MKL. Random dense matrices are used for the experiments, and the average execution time and speedup are examined.

Kato and Hosino [2010b] propose another parallel version of SVD on the GPU implemented in CUDA. The order of the computations of the U and V matrices is altered. Instead of examining all of the input data step by step, when the element a_{ij} of the sparse matrix A that contains the ratings is processed, the i-th row of U and the j-th row of V are updated. This means that all rows of U can be updated in parallel. First, U is updated for each $a_{ij} \neq 0$ and then V. The results are compared to those of a single-threaded implementation on a modern CPU. The time needed for one step of the iteration of convergence is measured.

Approximate SVD is parallelized by Chua [2012] using the R and C languages and the CUDA architecture. A single-node GPU kernel and a distributed GPU kernel over six nodes are used to approximate the matrix A, which contains the ratings. The algorithm is parallelized following the description of Kato and Hosino [2010b]. The total execution time and computations versus communication time are given. However, the author reports that the performance of the implemented algorithm is very sensitive to changes in the learning parameters, and the implementation only works for square matrices of sizes up to 1,024.

Approximate SVD using CUDA is also addressed by Foster et al. [2012]. The proposed method is based on an SVD method, called *QUIC-SVD* [Holmes et al. 2008], which is an approximate SVD algorithm that utilizes a tree-based structure. The algorithm is implemented on the CUDA architecture with the CULA library for linear algebra. Measures have been taken to be able to process matrices with a size larger than that of the GPU or main memory. The ratings matrix is divided in submatrices, and QUIC-SVD runs on every submatrix. Blocks of the ratings matrix are loaded into memory and are sequentially processed. A cosine tree is created for each submatrix, and a common basis is shared among the trees. The algorithm's results are compared to those of a multithreaded CPU version and two other implementations of SVD. Random matrices of various sizes are used for the experiments, and running time and speedup are provided.

Hansen et al. [2011] describes the parallelization of the nonparametric co-clustering model on a GPU. In this implementation, computations are made on both the CPU and the GPU. The speedup of the GPU computations over the CPU computations is measured. Two datasets are used for the collaborative filtering domain: the Netflix dataset and a Facebook dataset of user application consumption.

The SGD algorithm is parallelized on a GPU by Zastrau and Edelkamp [2012]. A hush function is created to facilitate executing threads in parallel. The implementation of the SGD algorithm is compared to an implementation of ALS on a GPU and to a batch gradient descent. The Netflix dataset is used, and the RMSE as well as execution time and scalability are measured.

One of the main reasons that restricted Boltzmann machines are often used in collaborative filtering is their property of being able to easily handle large datasets

[Salakhutdinov et al. 2007]. A preference has recently been shown for the usage of restricted Boltzmann machines for collaborative filtering algorithms on GPUs. A restricted Boltzmann machine is applied to collaborative filtering by Cai et al. [2012], and a parallel implementation on GPUs using CUDA is discussed. The computations of the collaborative filtering RBM are remodeled as matrix operations to be implemented in CUDA. The Java programming language and the JCUDA library are used. The experiments are run on the Netflix dataset, and the implementations speedup is examined.

The same authors also applied restricted Boltzmann machines on GPUs [Cai et al. 2013]. The matrix multiplications on GPUs described in their previous work [Cai et al. 2012] are adjusted to a hybrid framework that schedules the use of CPUs and GPUs. A CPU thread controls the scheduler, and another thread activates the CUDA kernels. The remainder of the CPU cores undertake the multiprocessor kernels. The framework is implemented in JAVA, and the JCUDA library is used for the CUDA kernels. The speedup of the hybrid implementation is compared to that of a CUDA implementation and to that of a multithreaded implementation. The runtime of the hybrid kernel is given, and the proportion of the CPU computation and hybrid kernel's runtime is discussed. Information about the used dataset is vague.

Asynchronous SGD based on Hogwild is implemented using GPUs by Noel and Osindero [2014]. The presented approach follows the master-worker model. The master is responsible for sending data to workers, and the workers compare the received data to a copy and update their local copy of data only once after the master has finished sending the data. Finally, the updates are sent to the master. Race conditions occur; however, as in Hogwild, they do not have significant impact to the accuracy.

The technologies and the datasets used by each model-based algorithm implementation on GPUs can be found in Table XIII.

It is observed that among the parallel implementations, the model-based algorithms are preferred. The applications on shared memory models represent the most recent applications, whereas algorithms continue to be developed on distributed memory systems. The development of recommendation algorithms on GPUs is also very active. The parallel implementations are mostly evaluated by the achieved speedup over the sequential versions. Although accuracy is often measured in both shared and distributed memory implementations, it seems that it is of less importance to the GPU-based implementations. This can be justified because the GPU-based approaches focus on the strategies that they use to process the data and not on the design of new algorithms. Thus, it would be interesting to see new or modified collaborative filtering algorithms implemented on GPUs.

In all parallel implementations, a clear preference for model-based algorithms is observed. As shown in Table VII, the majority of the algorithms implemented on distributed memory systems are model-based algorithms. Only one hybrid approach can be found, and none of the approaches implements memory-based algorithms.

The algorithms that are more often implemented on distributed memory systems are ALS, SGD, and co-clustering methods. MPI is used for communication among the system's nodes in all implementations. For these approaches, speedup is the metric that is most often used for evaluation. The Netflix dataset is used for almost all implementations, followed by the MovieLens and KDD Cup 2011 dataset.

The shared memory collaborative filtering implementations are listed according to their publication year in Table IX. Although the number of shared memory approaches is not sufficient to draw significant conclusions, one interesting fact is that all approaches are very recent. A preference to model-based approaches is shown, without indicating any inclination as to a specific algorithm. The Netflix, MovieLens, and Yahoo!

Table XIII. Model-Based Implementations on the GPU

Algorithm	Technologies	Datasets	Metrics
SVD [Vinay et al. 2006]	CUDA Intel MKL	N/A	Time for bidiagonalization
SVD [Lahabar and Narayanan 2009]	CUDA CUBLAS library Matlab	Random dense matrices	Average execution time Speedup
SVD [Kato and Hosino 2010b]	CUDA	Random data	Time for one step of the iteration of convergence
Nonparametric co-clustering [Hansen et al. 2011]	CUDA	Netflix Facebook	Speedup AUC
Approximate SVD [Chua 2012]	R,C CUDA	N/A	Total execution time Computation/communication time
RBM for collaborative filtering (CF) [Cai et al. 2012]	CUDA, Java JCUDA library	Netflix	Speedup
SGD [Zastrau and Edelkamp 2012]	CUDA	Netflix	RMSE, execution time Speedup
Approximate SVD QUIC-SVD [Foster et al. 2012]	CUDA CULA library	Random matrices	Running time Speedup
RBM for CF [Cai et al. 2013]	CUDA, Java JCUDA library	Self-generated	Speedup Runtime
Dogwild [Noel and Osindero 2014]	CUDA	Streaming data	Time (sec) accuracy

Music datasets are used to conduct experiments. For the most recent implementations, all three datasets are used to provide more accurate explanations of the results.

Time-related measurements seem to be more important in the shared memory implementations than in the distributed memory implementations, having speedup and scalability analyzed in almost all implementations. The RMSE metric is also considered by the majority of the implementations, whereas none of the implementations conduct experiments using the MAE metric. Furthermore, it is important to observe that none of the shared memory implementations combine model- and memory-based algorithms.

All implementations developed using GPUs are built on CUDA. A preference to the model-based algorithms is also shown in the implementations that utilize GPU accelerators. Most of the memory-based applications parallelize the user-based algorithm, and a few address the item- and neighborhood-based algorithms. However, the memory-based implementations are too few in number to allow for sufficient conclusions. The preferred datasets are the Flixster and MovieLens datasets.

Regarding the metrics, a preference is noticed for the measurement of the total execution time and the speedup over the sequential implementations. RMSE is seemingly less important to researchers because no attention is given to proving the selected algorithms' efficiencies. They are more concerned with comparing the CPU and GPU execution times. For the first time, a focus on the power and energy consumption of the implementations is noticed. Although this metric is utilized in only one implementation, other works are also expected to concern such issues in the future.

Among the model-based collaborative filtering algorithms, the algorithm that has been consistently selected for parallelization on GPUs is SVD. Other algorithms, such as SGD, co-clustering, and the usage of restricted Boltzmann machines on collaborative

filtering, have also been implemented, although not to such an extent. The majority of the model-based algorithms that have been implemented using CUDA employ libraries, such as CUBLAS or CULA, to handle the various algebraic problems that they encounter more efficiently.

It is interesting that many of the model-based approaches on GPUs select random datasets for the experiments. This fact, in addition to negatively affecting the experimental reproducibility, also complicates the comparison of the results to those of other implementations. Aside from randomly produced datasets, the Netflix dataset is the most used. The metrics that are mostly preferred are the speedup and execution time. Measuring computation and communication time, as well as RMSE, rarely occurs. When using very large datasets on GPUs, the problem of high data transfer times between CPUs and GPUs occurs and can significantly affect the overall performance. Fortunately, major companies in the field have recently announced developments of new technologies that can address this challenge via unified memory [Developer Zone NVIDIA 2013; Developer Central AMD 2014]. Consequently, GPUs are expected to be used more extensively for the development of applications that will utilize the information provided by very large datasets.

5. PLATFORM-BASED RECOMMENDATIONS

Because the available amount of data is continuously increasing, it is inevitable that new methods for facilitating and expediting its elaboration will be developed. To this end, the use of Big Data frameworks represents a significant contribution. This section is devoted to the implementations of collaborative filtering recommendation algorithms realized with the aid of frameworks that are adequate for parallel processing and for the handling of large datasets. The implementations will be classified into memory- and model-based implementations, and they will be discussed according to their publication year, commencing with the oldest one. Table XIV lists the implementations that are based on frameworks.

The field opens with a hybrid approach that provides recommendations to Google News users [Das et al. 2007]. The model-based PLSI and MinHash clustering algorithms are combined with the item co-visitation counts. The MapReduce framework is used to parallelize the MinHash clustering method and the expectation maximization (EM) algorithm. The user's click history constitutes the input of the algorithm's map phase, which is conducted over various machines. The algorithm outputs key-value pairs that correspond to the clusters to which each user belongs. A comparison of the MinHash and PLSI algorithms proves that their combination performs worse than the original algorithms. Information on the used datasets and the metrics selected for evaluation are provided in Table XV. In Table XVI, the datasets and metrics used in each memory-based implementation can be found, and Table XVII presents information on the model-based implementations.

5.1. Memory-Based Implementations

Zhao and Shang [2010] implement a user-based collaborative filtering algorithm following the MapReduce model on the Hadoop platform. The algorithm is divided into three phases. In the data partitioning phase, the user IDs are separated into different files and are used as input during the map phase. The algorithm includes a map phase, where the recommendation list for each user is calculated, and a reduce phase, where all information calculated is collected, and output is generated. The algorithm's speedup is considered on the Netflix dataset.

A parallel user profiling approach is proposed by Liang et al. [2010]. The suggested implementation is developed on the Hadoop Map-Reduce framework and Cascading [Cascading 2009], therein using the Del.icio.us dataset [Delicious 1998] on the Amazon

Table XIV. List of Implementations on Frameworks

Reference	Category	Description
Das et al. [2007]	HYBRID	MinHash and PLSI clustering Co-visitation counts
Chen et al. [2009]	MODEL	LDA
Wang et al. [2009]	MODEL	PLDA
Daruru et al. [2009]	MODEL	Bregman co-clustering
Zhao and Shang [2010]	MEMORY	User based
Liang et al. [2010]	MEMORY	User profiling
Davidson et al. [2010]	MEMORY	Distributed item based
Zinkevich et al. [2010]	MODEL	SGD
Jiang et al. [2011]	MEMORY	Item based
Gemulla et al. [2011]	MODEL	DSGD
Ali et al. [2011]	MODEL	Distributed SGD
Wu et al. [2011]	MEMORY AND MODEL	Collaborative filtering library: ALS, Wals, BPTF, SGD, SVD++, Item-kNN, Time-kNN, Time-SGD, Time-SVD++, MFITR
Chen and Hongfa [2011]	MEMORY	User-based clustering Slope one (CWSO)
Schelter et al. [2012]	MEMORY	Pairwise item comparison Top-N recommendation
Kanagal et al. [2012]	MODEL	Taxonomy-aware latent factor
Schelter et al. [2013]	MODEL	ALS
Tang and Harrington [2013]	MODEL	Truncated SVD and ALS
Li et al. [2013]	MODEL	DSGD
Sparks et al. [2013]	MODEL	ALS, SGD
Du et al. [2014]	MODEL	Maxios (NNMF-ADMM)
Johnson [2014]	MODEL	Logistic-MF
Udell et al. [2014]	MODEL	PCA
Xu et al. [2014]	MEMORY	DSingCF (item based)

Table XV. Hybrid Implementations on Frameworks

Algorithm	Technologies	Datasets	Metrics
MinHash clustering EM, PLSI [Das et al. 2007]	MapReduce	MovieLens, Google News	Precision, recall, live traffic ratios

EC2 EMR clouds. To create the user profiles, a tag vector is formed for each user. The recommendation is obtained by the user-based algorithm, therein using the cosine similarity to select the k-nearest neighbors. The top-N items are recommended according to the prediction value. Three Cascading flows implement the user profiling phase, the formation of the neighborhood and the recommendation phase. A comparison is given of the running time for the three jobs on the cloud and on a local desktop machine.

Personalized video recommendations are made through YouTube's distributed item-based recommendation system [Davidson et al. 2010]. Item similarity is calculated considering the user's co-visitation counts. To process large amounts of data, recommendations are calculated following a batch-oriented precomputation approach of MapReduce computations. The recommendation quality is evaluated through the following metrics: click-through rate (CTR), long CTR, session length, time until first

Table XVI. Memory-Based Implementations on Frameworks

Algorithm	Technologies	Datasets	Metrics
User-based collaborative filtering (CF) [Zhao and Shang 2010]	MapReduce Hadoop	Netflix	Speedup
Parallel user profiling [Liang et al. 2010]	MapReduce Hadoop	Del.icio.us	Running time
Distributed item-based YouTube's Recommender System [Davidson et al. 2010]	MapReduce BigTable	Live Traffic (self-collected)	CTR Long CTR Session length Recommendation coverage Time until first long watch
Item-based CF [Jiang et al. 2011]	MapReduce Hadoop	MovieLens	Isoefficiency, speedup
CF library (GraphLab) item-KNN, time-KNN [Wu et al. 2011]	GraphLab	Yahoo! Music	RMSE, speedup
User-based clustering weighted slope one (CWSO) [Chen and Hongfa 2011]	Hadoop Weka	MovieLens	MAE, acccuracy
Pairwise item comparison and top-N recommendation [Schelter et al. 2012]	MapReduce Hadoop	MovieLens Flixter Yahoo! Music	MAE, RMSE speedup, runtime
DSingCF [Xu et al. 2014]	MapReduce Hadoop	MovieLens	Speedup, execution time MAE, RMSE, NDCG

long watch, and recommendation coverage. Unfortunately, no other implementations assume these metrics.

The item-based collaborative filtering algorithm is implemented on Hadoop by Jiang et al. [2011]. This approach separates the three most excessive computations into four map-reduce phases, which are executed in parallel on a three-node Hadoop cluster. In the first map-reduce phase, the average rating for each item is computed. The second map-reduce phase computes the similarity between item pairs, and in the third map-reduce phase, the similarity matrix is recorded. Finally, the computations for the item prediction occurs in the fourth map-reduce phase. The MovieLens dataset is used, and isoefficiency and speedup scalability metrics are used to measure the implementation's performance.

Chen and Hongfa [2011] implement a user-based clustering weighted slope one (CWSO) algorithm using Hadoop on a five-machine cluster. This approach clusters users and assigns weights to each cluster. Then the ratings are predicted using weighted slope one. The prediction is accomplished with two map-reduce phases. In the first phase, a list of the items that are rated and belong to the same cluster as the active user's cluster is constructed. In the second phase, the average deviation between two items is calculated and used for the prediction. Users are clustered with the k-means algorithm on WEKA [University of Waikato 1997]. The MovieLens dataset is used, and MAE and accuracy are measured.

A neighborhood-based algorithm for batch recommendation is implemented on the Hadoop MapReduce framework by Schelter et al. [2012]. One map-reduce phase counts the item co-occurrences without considering the rating values that have been given to the items. The item vectors are preprocessed to compute their norm and their dot products and finally proceed to the similarity computation. Another map-reduce phase applies a threshold to sparsify the similarity matrix, therein omitting very low

Table XVII. Model-Based Implementations on Frameworks

Algorithm	Technologies	Datasets	Metrics
Parallel LDA [Chen et al. 2009]	MPI MapReduce	Orkut	Scalability, speedup, running time
PLDA [Wang et al. 2009]	MPI MapReduce	Wikipedia A forum dataset	Speedup Computation time Communication time Running time
Co-clustering Dataflow Bregman [Daruru et al. 2009]	Pervasive DataRush library	Netflix	RMSE, speedup Prediction/training time
SGD [Zinkevich et al. 2010]	MapReduce	Email system	RMSE
Distributed stratified DSGD [Gemulla et al. 2011]	R and C, Hadoop	Netflix	Speedup, elapsed wall-clock time
Distributed SGD (streaming data) [Ali et al. 2011]	MapReduce Hadoop, Storm	MovieLens	Total elapsed time vs. RMSE, number of iterations vs. RMSE
Collaborative filtering library (GraphLab) ALS, Wals, BPTF, SGD, SVD++, time-SGD, time-SVD++, MFITR, time-MFITR [Wu et al. 2011]	GraphLab	Yahoo! Music	RMSE, speedup
Multicore (TF) taxonomy-aware latent factor model (SGD) [Kanagal et al. 2012]	C++ BOOST library Hadoop	A log of user online transactions	AUC, speedup Absolute wall-clock time Average mean rank of users
ALS [Schelter et al. 2013]	MapReduce Hadoop JBlas	Netflix Yahoo! Music Bigflix (synthetic)	Average runtime per recomputation
Truncated SVD, ALS [Tang and Harrington 2013]	MapReduce	Collected from Walmart.com	MAP, NDCG
DSGD [Li et al. 2013]	Spark, Sparkler, Scala	Netflix	Execution time per epoch
ALS, SGD [Sparks et al. 2013]	Spark, Scala, Matlab GraphLab, Mahout, Matlab-Mex	Netflix	Scalability Execution time
Maxios [Du et al. 2014]	Spark, Scala Matlab	MovieLens 1M, Netflix Yahoo! Music	Scalability RMSE vs. time
Logistic MF [Johnson 2014]	MapReduce	Spotify data	Mean percentage ranking (MPR)
PCA [Udell et al. 2014]	Spark Scala	Netflix	Iteration time (sec)

similarities. Batch recommendation can be completed in a map-only phase if the similarity matrix fits into the memory. Otherwise, a reduce phase is used. To reduce the algorithm's cost, which is dominated by "power users," only a randomly selected part of their interactions contributes to the recommendation. Sensitivity analysis is given for the effects of the user interaction reduction. Both the MovieLens and Flixster datasets are used for measuring the algorithm's accuracy. Furthermore, the algorithm is evaluated based on scalability and runtime on the Yahoo! Music dataset.

DSingCF [Xu et al. 2014] is a two-phase algorithm, which in addition to using only item pairs uses unique item ratings in the recommendation procedure. During the first phase, the missing ratings are estimated such that all possible item pairs can have a similarity value. The second phase is dedicated to finding the most similar users. Two map-reduce phases are executed during the first phase: one for the calculation of each user's average rating and one for the calculation of the missing ratings. The second phase of the algorithm is completed with one map-reduce phase.

5.2. Model-Based Implementations

A parallel version of the LDA algorithm is presented by Chen et al. [2009]. LDA's parallelization is accomplished with the MPI library and MapReduce. Subsets of the users and their ratings are divided among the available machines. The communication and the synchronization among the processes are accomplished with MPI, whereas map and reduce functions are defined with the MapReduce framework. A detailed description of the MPI-based PLDA algorithm and a version of MapReduce are given by Wang et al. [2009]. The MPI implementation is publicly available; thus, the experiments can be reproduced.

The only implementation that utilizes the Pervasive DataRush library [Actian DataRush 2009] develops a parallel implementation of the Bregman co-clustering algorithm [Daruru et al. 2009]. Both co-clustering training and prediction algorithms are implemented using a dataflow graph. The Pervasive DataRush library is used to construct and execute the dataflow graphs. The number of used cores influences the number of data partitions that will be processed. An evaluation is provided, and a few optimizations, such as the use of JOMP or adjusting the distance computations according to a technique better suited for sparse data, are proposed.

A parallel SGD algorithm for MapReduce is described by Zinkevich et al. [2010]. In the presented method, SGD runs in parallel on different computers, and their results are aggregated. The only communication needed between the computers is during the collection of the results; thus, only one map-reduce phase is needed. The experiments are run on a dataset formed by an email system, and the results are evaluated based on the RMSE.

Gemulla et al. [2011] also develop a stratified variant of SGD that is adjusted to obtain the distributed algorithm DSGD. The input data are distributed over the nodes at the beginning of the execution, and smaller matrices are transmitted during the remainder of the execution. Each node creates a local training sequence from the data that it receives. During each iteration, a step size and a stratum are chosen. Then SGD runs on the training points in such a way that the entire training set is finally covered. For the experiments, two clusters are used: a cluster of 32 cores for the in-memory implementation, which is based on R and C, and a Hadoop cluster consisting of 320 cores. The Netflix dataset is used, and speedup and the elapsed wall-clock time are measured.

An extension of the preceding SGD algorithm is presented by Ali et al. [2011]. This approach is designed to operate on streaming data and is implemented on a cluster composed of 10 machines using the Hadoop Map-Reduce and Storm frameworks. The master node dynamically assigns data chunks to workers, therein taking care to avoid

the need for the simultaneous update of the same rows or columns. To compute a stratum, the input to the map phase is the ratings matrix and the U and M matrices. If the rating belongs to the current stratum, the mapper outputs the key-value pairs that correspond the stratum blocks to the ratings that they contain. The reducers receive the information that belongs to a stratum block, and SGD runs on them. The MovieLens dataset is used, and the results are presented on plots of the total elapsed time versus RMSE and of the number of iterations versus RMSE.

Wu et al. [2011] implement an open-source collaborative filtering library using the GraphLab parallel machine learning framework. The implemented algorithms are ALS, Wals, BPTF, SGD, SVD++, Item-kNN, time-kNN, time-SGD, time-SVD++, MFITR, and time-MFITR. Although a few memory-based algorithms are implemented, emphasis is given to the matrix factorization algorithms. Experiments are conducted on a cluster composed of 32 cores and on the BlackLight supercomputer [Pittsburgh Supercomputing Center 2009] (4,096 cores). The RMSE metric is measured on the validation dataset, and the speedup is calculated on BlackLight. The Yahoo! Music dataset is used.

Kanagal et al. [2012] develop a parallel multicore implementation of the taxonomy-aware latent factor model (TF), implemented in C++. The BOOST library is also used. The SGD algorithm is applied via a multithreaded implementation. Using Hadoop, a different part of the set of users is assigned to each node. The input data is a log of user online transactions. The AUC metric and the average mean rank of the users are used to compare the proposed model with the basic latent factor model. In addition, absolute wall-clock times and speedup are measured on a 12-core machine.

Schelter et al. [2013] parallelize the ALS algorithm on MapReduce using a parallel broadcast-join. Each machine has a local part of the matrix A that contains the user's interactions over the set of items. Furthermore, the smaller of the feature matrices U and M that contain the user and item information, respectively, is replicated to all available machines. A map phase joins the local part of A and the replicated copy of the feature matrix and recomputes the other feature matrix. The experiments are realized using three datasets: Netflix, Yahoo! Music, and Bigflix, which is a synthetic dataset constructed from the Netflix dataset. The average runtime for a recomputation of the feature matrix is measured.

Tang and Harrington [2013] propose a two-stage matrix factorization. The truncated SVD algorithm is first run on a MapReduce cluster. Then the ALS algorithm is applied, starting with the matrix that has been received as a result of the truncated SVD instead of using a random matrix Q. The matrix P is calculated with one map-reduce step. To evaluate this approach, two metrics are used: mean average precision (MAP) and normalized discounted cumulative gain (NDCG). Unfortunately, these metrics are not used in other similar experiments, and no information is given on whether the data collected from the Walmart.com site can be made publicly available.

Li et al. [2013] implement DSGD on Spark [Zaharia et al. 2012] and on an extension of Spark called *Sparkler*. Spark is a distributed in-memory framework applicable to iterative methods. Sparkler is adequate for matrix factorization using a distributed memory abstraction for large factors, called *Carousel Map*. Using Spark, each node has a copy of the latent factor matrices. The ratings are separated into blocks that belong to stratums, and the blocks of the stratums are distributed over the nodes. The blocks of each stratum are processed in parallel, and each node updates the corresponding parts of the factor matrices. DSGD on Sparkler differs from this approach. Carousel Map is used to process the data that are distributed over the nodes and to handle the communication among the nodes. Li et al. [2013] compare the performance of the DSGD algorithm on Spark and Sparkler.

Sparks et al. [2013] present an application called *MLI* for distributed machine learning algorithms built on Spark. To evaluate their application, they develop two algorithms. The performance of SGD and ALS on MLI is compared to that of Matlab versions.

Du et al. [2014] propose a weighted nonnegative matrix factorization algorithm for large sparse matrices, Maxios, implemented on Spark. The computation time of Maxios is reduced by calculating the missing values only once, after all updates have been completed, and by using alternating direction methods of multipliers (ADMM) to update the multipliers in each iteration. Each worker node has a copy of the rows and columns of the latent factor matrices and the ratings matrix. In each iteration, the latent factors are broadcast to the worker nodes.

Johnson [2014] presents a probabilistic matrix factorization model for music recommendations called *Logistic MF*. The behavior of the users toward the items and the popularity of the items are expressed in the model via biases. The summations needed to compute the gradients of each iteration are calculated in parallel using MapReduce. The input matrix is partitioned into submatrices. Each mapper receives the user and item vectors of a block that corresponds to a specific user and item and updates the gradients. During the reduce phase, users or items are used as keys according to whether the performed iteration is a user or an item iteration, and the summations are aggregated.

Low-rank matrix approximation is addressed by Udell et al. [2014], who use Spark to develop a distributed implementation of PCA. Each machine has a part of the data, and the cores of each machine process different parts of the data. Communication occurs at each iteration. The implementation has been designed for models that fit in memory.

High activity in the development of collaborative filtering algorithms using platforms has been observed recently. A preference for the development of model-based algorithms is noticed, and most of the implementations are developed on Hadoop and follow the MapReduce model. Interest in the development of model-based algorithms using the Spark framework has also been observed recently. In platform-based approaches, both scalability and accuracy are evaluated.

Among the platform-based implementations, only one hybrid implementation combining both model- and memory-based techniques is observed. In addition, there is no definite trend in favor of one of the two categories. Both model- and memory-based algorithms have been chosen for implementation on frameworks. The algorithms more often employed among the model-based implementations on frameworks are LDA, SGD, and SVD, and among the memory-based algorithms, user-based and item-based collaborative filtering are most often employed.

Various datasets are used to evaluate the discussed approaches. Although many approaches are evaluated on a variety of datasets, the dominating datasets are Netflix, MovieLens, and Yahoo! Music. The majority of the applications are implemented on the Hadoop MapReduce framework; however, especially for model-based implementations, various approaches that combine Hadoop with other parallel computing libraries, such as MPI or Pervasive DataRush, have been developed. In addition, many algorithms have been implemented on GraphLab and Spark.

Concerning the most commonly used metrics, a preference is noticed for RMSE, MAE, running time, and speedup. The fact that the implementations have been executed on systems that significantly differ in the number and specifications of used cores proves that it is difficult to attain an overall comparison.

6. HETEROGENEOUS IMPLEMENTATIONS

A few hybrid collaborative filtering implementations have recently been developed on both shared and distributed memory systems. Most of these implementations have

Table XVIII. List of Heterogeneous Implementations

Reference	Category	Description
Narang et al. [2011b]	MODEL	Co-clustering
Narang et al. [2011a]	MODEL	Co-clustering
Karydi and Margaritis [2012b]	MEMORY	Slope one
Guan et al. [2012]	MEMORY	Semisparse multilayer optimization on item based
Schelter et al. [2014]	MODEL	Factorbird

Table XIX. Heterogeneous Implementations

Algorithm	Technologies	Datasets	Metrics
Distributed co-clustering [Narang et al. 2011b]	MPI, OpenMP	Netflix	RMSE, scalability Training time Prediction time per rating
Distributed co-clustering variations [Narang et al. 2011a]	MPI, OpenMP	Netflix Yahoo KDD Cup	(Weak, strong, data) Scalability RMSE
Slope one [Karydi and Margaritis 2012b]	MPI, OpenMP	MovieLens	Scalability, speedup Total elapsed time Prediction per second Prediction time per rating
Semisparse multilayer optimization (item based) [Guan et al. 2012]	MPI Pthreads	MovieLens Netflix	Speedup Elapsed CPU time
Factorbird [Schelter et al. 2014]	Scala, HDFS, MapReduce	Subset of Twitter graph	RMSE

been performed with MPI and OpenMP or Pthreads. For the remainder of this section, they will be described starting with the oldest and proceeding to the most recent implementation. Table XVIII presents a list of these approaches. The datasets used in each implementation, as well as the metrics considered, can be found in Table XIX.

A distributed model–based algorithm based on co-clustering is presented by Narang et al. [2011b]. The algorithm partitions row and column clusters into the nodes, which are further partitioned into each node's threads. Iterations are executed until reaching the desired RMSE convergence. One thread on each node, apart from contributing to the computations, takes over the necessary communication to collect the results of the computations assumed by the remaining threads. The Netflix Prize dataset is used on a 1,024-node Blue Gene/P architecture. Training and prediction times as well as the RMSE are measured. A detailed scalability analysis is also presented.

Other variations of the distributed co-clustering–based collaborative filtering algorithm are presented by Narang et al. [2011a]. A distributed flat co-clustering algorithm is implemented using MPI, and a flat hybrid algorithm is developed using MPI and OpenMP. Hierarchical co-clustering algorithms are also developed. The algorithms are evaluated on the Blue Gene/P architecture, and the utilized datasets and metrics can be found later in Table XIX.

Karydi and Margaritis [2012b] present a hybrid version of the slope one algorithm and compare its performance to that of the multithreaded version, which is described in Karydi and Margaritis [2012a]. Parts of the ratings matrix are distributed over the system's nodes. The master-worker model is followed. The master node assumes the data partitioning and distribution, whereas the worker nodes are devoted to the

computations. Finally, all workers' results are gathered and sent to the master node, where the predictions are made. This implementation is evaluated on a heterogeneous cluster composed of 30 cores and a homogeneous cluster composed of 24 cores. The MovieLens dataset is used for the performance and scalability evaluation, and total elapsed time, speedup, number of predictions per second, and prediction time per rating are measured.

A semisparse algorithm that attempts to accelerate the common memory-based collaborative filtering algorithms is proposed by Guan et al. [2012]. Three optimization methods are applied. First, a semisparse algorithm that locally concentrates the selected sparse vectors is used to speed up the similarity computations. On a multicore architecture, threads are wrapped into a thread pool, and a reduced vector is used to reduce the use of locks. Moreover, to reduce the communication overhead among different nodes, a shared zip file that contains the sparse rating matrix is read. Experiments are conducted on three different multicore systems and on a cluster of eight nodes.

Schelter et al. [2014] present an algorithm based on SGD. Factorbird is based on a parameter server architecture [Li et al. 2014]. A part of the machines is used to learn the model, and another part is used as parameter servers that hold the parts of the factor matrices. The data are partitioned over the learner machines, and each machine runs a multithreaded SGD. To avoid conflicting updates, the algorithm is based on Hogwild [Recht et al. 2011]. A cluster is used for the parameter machines, and Scala [Odersky et al. 2004] and the HDFS file system are used for the learner machines. The data are a subset of Twitter's interaction graph, and the algorithm is evaluated based on the RMSE.

Heterogeneous implementations that combine several parallelization techniques are very few in number, and mainly, distributed memory approaches are combined with shared memory models. More heterogeneous implementations can be developed, therein combining various parallelization techniques.

Unfortunately, the results of the heterogeneous implementations cannot be compared to each other, not only because of the use of different datasets but also because of the use of different cluster architectures with significant difference in the number of nodes. No preference is given to either model-based or memory-based algorithms. In all of these implementations, the communication among the cluster nodes is accomplished via MPI, and OpenMP or Pthreads is used for shared memory parallelization. The dominating datasets are the Netflix and MovieLens datasets. For these implementations, priority is given to measuring their scalability and speedup, whereas measuring the algorithm's accuracy via the RMSE metric is of less interest.

Among the memory-based algorithms, user-based algorithms are implemented more often, followed by item-based algorithms. The most frequently implemented model-based algorithms are SVD, SGD, ALS, and co-clustering models. Among the implementations of hybrid algorithms, no main scheme can be distinguished.

7. DISCUSSION AND CONCLUSIONS

Because research works concerning recommender systems are published in a variety of journals that focus on different disciplines [Park et al. 2012] and because the field of recommendation algorithms exhibits high activity, it is not easy to ensure that all existing implementations are considered in this survey. Regardless, no changes to the conclusions drawn from this work are expected if a few more works appear.

An initial observation is that regardless of the parallel or distributed method used, fewer hybrid implementations exist compared to memory- and model-based implementations. Hence, additional hybrid algorithms could be developed that would benefit from both categories' advantages. Another fact worth mentioning is that no memory-based implementations have been developed on distributed memory systems, and only

one has been developed on a shared memory environment. However, because memory-based collaborative filtering algorithms also deliver good results, they should not be ignored.

The present work demonstrates that the field of parallel and distributed collaborative filtering is active and evolving quickly. We have attempted to catalog as many implementations that have been published in scientific journals or conferences before the end of 2015 as possible. Furthermore, a category of works that make slight use of recommender systems but whose main focus is on neural networks or other artificial intelligence techniques has been omitted from the present work.

Recently, many parallel and distributed collaborative filtering approaches have been developed, particularly in employing GPUs or taking advantage of various platforms. However, more research needs to be conducted to exploit the benefits of parallel and distributed computing techniques and improve the collaborative filtering algorithms in such a way so as to handle the huge amounts of data that are available more efficiently.

It would be interesting to apply a multilevel heterogeneous method, using many machines to efficiently handle big data and subsequently combine a variety of techniques according to the addressed problem. In recent years, although a variety of parallel and distributed techniques have been applied, a preference is noticed for the use of graphics accelerators and frameworks. Thus, the use of an adequate framework in combination with MPI and GPU accelerators would be intriguing. One aspect that is determinant for the selection of a technique is the nature of the available data. If it is difficult to collect all data in one machine, then distributed methods should be preferred, whereas clusters or methods based on shared memory environments are better suited for when data are easily assembled in one place.

Briefly summarizing the findings of the research work discussed in this article, the preference of the research community for the development of model-based collaborative filtering algorithms is clear. Memory-based and especially hybrid algorithms are implemented less often. Still, the development of hybrid algorithms seems promising because both methods' benefits could be utilized.

In recent years, a main trend for the use of frameworks and GPU accelerators has been noticed, with MPI-based and shared memory techniques in second place. The use of frameworks is anticipated to be more flexible in the future and to be combined with other techniques.

As far as the evaluation of the implementations is concerned, initially, algorithmic accuracy, which was measured using the MAE metric, was the main interest. Lately, the interest has turned toward scalability analysis and the achievement of lower execution times. A few approaches are tested on self-collected data, which are not publicly available for further experiments. However, the majority of the implementations are tested on the well-known Netflix, MovieLens, and Yahoo! Music datasets.

Despite the fact that many algorithms are sufficiently accurate, the availability of multiple data sources, such as social networks and the variety of possible items to recommend, is motivating researchers toward constantly improving existing algorithms and designing new algorithms such that more information can be utilized in the recommendation procedure. Therefore, the design of recommendation algorithms for parallel and distributed systems that can take advantage of social information or other implicit data could significantly advance the field.

Another open issue in the field of parallel and distributed recommendation algorithms is the development of algorithms that can be scalable when running on a variety of systems. Adaptive algorithms that will be able to recognize the system's available hardware and utilize the provided heterogeneity could represent a great solution toward providing portable code for heterogeneous systems.

As a conclusion, new technologies are continuously contributing to the development of parallel and distributed collaborative filtering algorithms. There is no specific pattern to be followed because the selection of the adequate technology is highly related to the nature of the available data, the characteristics of the algorithms, and the available hardware. The work discussed in this article is expected to provide a useful starting basis to offer helpful directions for both the selection oftechnologies and algorithms and to inspire the development of more sophisticated approaches.

REFERENCES

Actian DataRush. 2009. Pervasive DataRush Knowledge Discovery Platform. (2009). Retrieved March 14, 2014, from https://www-304.ibm.com/partnerworld/gsd/showimage.do?id=27569.

Gediminas Adomavicius and Alexander Tuzhilin. 2005. Toward the next generation of recommender systems: A survey of the state-of-the-art and possible extensions. *IEEE Transactions on Knowledge and Data Engineering* 17, 6, 734–749. DOI:http://dx.doi.org/10.1109/TKDE.2005.99

Jae-Wook Ahn and Xavier Amatriain. 2010. Towards fully distributed and privacy-preserving recommendations via expert collaborative filtering and restful linked data. In *Proceedings of the 2010 IEEE/WIC/ACM International Conference on Web Intelligence and Intelligent Agent Technology—Volume 01 (WI-IAT'10)*. IEEE, Los Alamitos, CA, 66–73. DOI:http://dx.doi.org/10.1109/WI-IAT.2010.53

Kamal Ali and Wijnand van Stam. 2004. TiVo: Making show recommendations using a distributed collaborative filtering architecture. In *Proceedings of the 10th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD'04)*. ACM, New York, NY, 394–401. DOI:http://dx.doi.org/10.1145/1014052.1014097

Muqeet Ali, Christopher C. Johnson, and Alex K. Tang. 2011. Parallel Collaborative Filtering for Streaming Data. Available at http://citeseerx.ist.psu.edu/index.

Dhoha Almazro, Ghadeer Shahatah, Lamia Albdulkarim, Mona Kherees, Romy Martinez, and William Nzoukou. 2010. A survey paper on recommender systems. arXiv:1006.5278.

Xavier Amatriain, Neal Lathia, Josep M. Pujol, Haewoon Kwak, and Nuria Oliver. 2009. The wisdom of the few: A collaborative filtering approach based on expert opinions from the Web. In *Proceedings of the 32nd International ACM SIGIR Conference on Research and Development in Information Retrieval (SIGIR'09)*. ACM, New York, NY, 532–539. DOI:http://dx.doi.org/10.1145/1571941.1572033

Apache Hadoop. 2016. Home Page. Retrieved July 18, 2016, from http://hadoop.apache.org/.

Apache Spark. 2009. Home Page. Retrieved July 18, 2016, from http://spark.apache.org/.

Apache Storm. 2012. Home Page. Retrieved July 18, 2016, from https://storm.apache.org/.

Audioscrobbler. 2002. Audioscrobbler Data. Retrieved October 10, 2012, from http://www.audioscrobbler.com.

Baruch Awerbuch, Boaz Patt-Shamir, David Peleg, and Mark R. Tuttle. 2005. Improved recommendation systems. In *Proceedings of the 16th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA'05)*. 1174–1183.

Arindam Banerjee, Inderjit Dhillon, Joydeep Ghosh, Srujana Merugu, and Dharmendra S. Modha. 2004. A generalized maximum entropy approach to Bregman co-clustering and matrix approximation. In *Proceedings of the 10th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD'04)*. 509–514. http://portal.acm.org/citation.cfm?doid=1014052.1014111

Alejandro Bellogín, Iván Cantador, and Pablo Castells. 2013a. A comparative study of heterogeneous item recommendations in social systems. *Information Sciences* 221, 142–169. DOI:http://dx.doi.org/10.1016/j.ins.2012.09.039

Alejandro Bellogín, Iván Cantador, Fernando Díez, Pablo Castells, and Enrique Chavarriaga. 2013b. An empirical comparison of social, collaborative filtering, and hybrid recommenders. *ACM Transactions on Intelligent Systems and Technology* 4, 1, Article No. 14. DOI:http://dx.doi.org/10.1145/2414425.2414439

Shlomo Berkovsky, Paolo Busetta, Yaniv Eytani, Tsvi Kuflik, and Francesco Ricci. 2005. Collaborative filtering over distributed environment. In *Proceedings of the DASUM Workshop*.

Shlomo Berkovsky, Yaniv Eytani, Tsvi Kuflik, and Francesco Ricci. 2007a. Enhancing privacy and preserving accuracy of a distributed collaborative filtering. In *Proceedings of the 2007 ACM Conference on Recommender Systems (RecSys'07)*. ACM, New York, NY, 9–16. DOI:http://dx.doi.org/10.1145/1297231.1297234

Shlomo Berkovsky and Tsvi Kuflik. 2006. Hierarchical neighborhood topology for privacy enhanced collaborative filtering. In *Proceedings of the CHI 2006 Workshop on Privacy-Enhanced Personalization (PEP'06)*. 6–13.

Shlomo Berkovsky, Tsvi Kuflik, and Francesco Ricci. 2007b. Distributed collaborative filtering with domain specialization. In *Proceedings of the 2007 ACM Conference on Recommender Systems (RecSys'07)*. ACM, New York, NY, 33–40. DOI: http://dx.doi.org/10.1145/1297231.1297238

Daniel Bernardes, Mamadou Diaby, Raphaël Fournier, Françoise FogelmanSoulié, and Emmanuel Viennet. 2015. A social formalism and survey for recommender systems. *SIGKDD Exploration Newsletter* 16, 2, 20–37. DOI: http://dx.doi.org/10.1145/2783702.2783705

J. Bobadilla, F. Ortega, A. Hernando, and A. GutiéRrez. 2013. Recommender systems survey. *Knowledge-Based Systems* 46, 109–132. DOI: http://dx.doi.org/10.1016/j.knosys.2013.03.012

Dheeraj Bokde, Sheetal Girase, and Debajyoti Mukhopadhyay. 2015. Matrix factorization model in collaborative filtering algorithms: A survey. *Procedia Computer Science* 49, 136–146. DOI: http://dx.doi.org/10.1016/j.procs.2015.04.237

Robin Burke. 2002. Hybrid recommender systems: Survey and experiments. *User Modeling and User-Adapted Interaction* 12, 4, 331–370. DOI: http://dx.doi.org/10.1023/A:1021240730564

Fidel Cacheda, Victor Carneiro, Diego Fernandez, and Vreixo Formoso. 2011. Comparison of collaborative filtering algorithms: Limitations of current techniques and proposals for scalable, high performance recommender systems. *ACM Transactions on the Web* 5, 1, Article No. 2.

Xianggao Cai, Zhanpeng Xu, Guoming Lai, Chengwei Wu, and Xiaola Lin. 2012. GPU-accelerated restricted Boltzmann machine for collaborative filtering. In *Proceedings of the 12th International Conference on Algorithms and Architectures for Parallel Processing—Volume Part I (ICA3PP'12)*. 303–316. DOI: http://dx.doi.org/10.1007/978-3-642-33078-0_22

Xianggao Cai, Zhanpeng Xu, Guoming Lai, Chengwei Wu, and Xiaola Lin. 2013. Design and implementation of large scale parallel collaborative filtering on multi-core CPU and GPU. Submitted to *Journal of Parallel and Distributed Computing*.

Laurent Candillier, Frank Meyer, and Marc Boullé. 2007. Comparing state-of-the-art collaborative filtering systems. In *Proceedings of the 5th International Conference on Machine Learning and Data Mining in Pattern Recognition (MLDM'07)*. 548–562. DOI: http://dx.doi.org/10.1007/978-3-540-73499-4_41

John Canny. 2002. Collaborative filtering with privacy. In *Proceedings of the 2002 IEEE Symposium on Security and Privacy (SP'02)*. IEEE, Los Alamitos, CA, 45. http://dl.acm.org/citation.cfm?id=829514.830525.

Cascading. 2009. Cascading: Big Data Application Development. Retrieved March 14, 2014, from http://www.cascading.org/.

Sylvain Castagnos and Anne Boyer. 2006. A client/server user-based collaborative filtering algorithm: Model and implementation. In *Proceedings of the 17th European Conference on Artificial Intelligence (ECAI'06)*. 617–621. http://dl.acm.org/citation.cfm?id=1567016.1567150

Sylvain Castagnos and Anne Boyer. 2007. Personalized communities in a distributed recommender system. In *Proceedings of the 29th European Conference on IR Research (ECIR'07)*. 343–355. http://dl.acm.org/citation.cfm?id=1763653.1763695

Wen-Yen Chen, Jon-Chyuan Chu, Junyi Luan, Hongjie Bai, Yi Wang, and Edward Y. Chang. 2009. Collaborative filtering for Orkut communities: Discovery of user latent behavior. In *Proceedings of the 18th International Conference on World Wide Web (WWW'09)*. ACM, New York, NY, 681–690. DOI: http://dx.doi.org/10.1145/1526709.1526801

Wen-Yen Chen, Dong Zhang, and Edward Y. Chang. 2008. Combinational collaborative filtering for personalized community recommendation. In *Proceedings of the 14th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD'08)*. ACM, New York, NY, 115–123. DOI: http://dx.doi.org/10.1145/1401890.1401909

X. Chen and W. Hongfa. 2011. Clustering weighted slope one for distributed parallel computing. *Computer Science and Network Technology* 3, 1595–1598.

Jack Chua. 2012. *Scaling Machine Learning Algorithms Across GPU Clusters Using R*. CSE 6220. Georgia Institute of Technology, Atlanta, GA.

George Coulouris, Jean Dollimore, Tim Kindberg, and Gordon Blair. 2011. *Distributed Systems: Concepts and Design* (5th ed.). Addison-Wesley.

Srivatsava Daruru, Nena M. Marin, Matt Walker, Joydeep Ghosh. 2009. Pervasive parallelism in data mining: Dataflow solution to co-clustering large and sparse Netflix data. In *Proceedings of the 15th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD'09)*. 1115–1123.

Abhinandan S. Das, Mayur Datar, Ashutosh Garg, and Shyam Rajaram. 2007. Google News personalization: Scalable online collaborative filtering. In *Proceedings of the 16th International Conference on World Wide Web (WWW'07)*. ACM, New York, NY, 271–280. DOI: http://dx.doi.org/10.1145/1242572.1242610

James Davidson, Benjamin Liebald, Junning Liu, Palash Nandy, Taylor Van Vleet, Ullas Gargi, Sujoy Gupta, Yu He, Mike Lambert, Blake Livingston, and Dasarathi Sampath. 2010. The YouTube video

recommendation system. In *Proceedings of the 4th ACM Conference on Recommender Systems (RecSys'10)*. ACM, New York, NY, 293–296. DOI:http://dx.doi.org/10.1145/1864708.1864770

Delicious. 1998. The Delicious Dataset. (1998). Retrieved June 17, 2013, from http://socialcomputing.asu.edu/datasets/Delicious.

Developer Central AMD. 2014. hUMA—The Next Big Thing in Processors. Available at http://developer.amd.com/.

Developer Zone NVIDIA. 2013. Unified Memory in CUDA 6. Retrieved July 18, 2016, from https://devblogs.nvidia.com/parallelforall/unified-memory-in-cuda-6/.

Simon Shaolei Du, Boyi Chen, Yilin Liu, and Lei Li. 2014. Maxios: Large scale nonnegative matrix factorization for collaborative filtering. In *Proceedings of the NIPS 2014 Workshop on Distributed Matrix Computations*.

Michael D. Ekstrand, John T. Riedl, and Joseph A. Konstan. 2011. Collaborative filtering recommender systems. *Foundations and Trends in Human-Computer Interaction* 4, 2, 81–173. DOI:http://dx.doi.org/10.1561/1100000009

Ignacio Fernandez-Tobias, Ivan Cantador, Marius Kaminskas, and Francesco Ricci. 2012. Cross-domain recommender systems: A survey of the state of the art. In *Proceedings of the 2nd Spanish Conference on Information Retrieval (CERI'12)*.

Blake Foster, Sridhar Mahadevan, and Rui Wang. 2012. A GPU-based approximate SVD algorithm. In *Proceedings of the 9th International Conference on Parallel Processing and Applied Mathematics—Volume Part I (PPAM'11)*. 569–578. DOI:http://dx.doi.org/10.1007/978-3-642-31464-3_58

Rainer Gemulla, Erik Nijkamp, Peter J. Haas, and Yannis Sismanis. 2011. Large-scale matrix factorization with distributed stochastic gradient descent. In *Proceedings of the 17th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD'11)*. 69–77.

T. George and S. Merugu. 2005. A scalable collaborative filtering framework based on co-clustering. In *Proceedings of the 5th IEEE International Conference on Data Mining (ICDM'05)*. 625–628.

GroupLens. 1997. MovieLens Datasets. Retrieved July 18, 2016, from http://www.grouplens.org/node/73.

Hu Guan, Huakang Li, and Minyi Guo. 2012. Semi-sparse algorithm based on multi-layer optimization for recommendation system. In *Proceedings of the 2012 International Workshop on Programming Models and Applications for Multicores and Manycores (PMAM'12)*. 148–155.

M. A. Hameed, O. A. Jadaan, and S. Ramachandram. 2012. Collaborative filtering based recommendation system: A survey. *International Journal on Computer Science and Engineering* 4, 5, 859–876. http://connection.ebscohost.com/c/articles/82397440/collaborative-filtering-based-recommendation-system-survey.

Peng Han, Bo Xie, Fan Yang, and Ruimin Shen. 2004a. A scalable P2P recommender system based on distributed collaborative filtering. *Expert Systems with Applications* 27, 2, 203–210. DOI:http://dx.doi.org/10.1016/j.eswa.2004.01.003

Peng Han, Bo Xie, Fan Yang, Jiajun Wang, and Ruimin Shen. 2004b. A novel distributed collaborative filtering algorithm and its implementation on P2P overlay network. In *Advances in Knowledge Discovery and Data Mining*. Lecture Notes in Computer Science, Vol. 3056. Springer, 106–115. DOI:http://dx.doi.org/10.1007/978-3-540-24775-3_13

T. J. Hansen, M. Morup, and L. K. Hansen. 2011. Non-parametric co-clustering of large scale sparse bipartite networks on the GPU. In *Proceedings of the 2011 IEEE International Workshop on Machine Learning for Signal Processing (MLSP'11)*. 1–6. DOI:http://dx.doi.org/10.1109/MLSP.2011.6064611

Andreas Harth, Michael Bauer, and Bernd Breutmann. 2001. *Collaborative Filtering in a Distributed Environment: An Agent-Based Approach*. In Technical Report. University of Applied Sciences, Würzburg, Germany.

Jonathan L. Herlocker, Joseph A. Konstan, Loren G. Terveen, and John T. Riedl. 2004. Evaluating collaborative filtering recommender systems. *ACM Transactions on Information Systems* 22, 1, 5–53.

Qirong Ho, James Cipar, Henggang Cui, Seunghak Lee, Jin Kyu Kim, Phillip B. Gibbons, Garth A. Gibson, Greg Ganger, and Eric P. Xing. 2013. More effective distributed ML via a stale synchronous parallel parameter server. In *Advances in Neural Information Processing Systems 26 (NIPS'13)*. 1223–1231. http://papers.nips.cc/paper/4894-more-effective-distributed-ml-via-a-stale-synchronous-parallel-parameter-server.pdf.

Thomas Hofmann. 2004. Latent semantic models for collaborative filtering. *ACM Transactions on Information Systems* 22, 1, 89115. http://portal.acm.org/citation.cfm?doid=963770.963774

M. Holmes, A. Gray, and C. L. Isbell. 2008. QUIC-SVD: Fast SVD using cosine trees. In *Advances in Neural Information Processing Systems 21 (NIPS'08)*. 673–680.

IMDb. 1990. Home Page. Retrieved July 18, 2016, from http://www.imdb.com.

Sibren Isaacman, Stratis Ioannidis, Augustin Chaintreau, and Margaret Martonosi. 2011. Distributed rating prediction in user generated content streams. In *Proceedings of the 5th ACM Conference on Recommender Systems (RecSys'11)*. ACM, New York, NY, 69–76. DOI: http://dx.doi.org/10.1145/2043932.2043948

Dietmar Jannach, Markus Zanker, Alexander Felfernig, and Gerhard Friedrich. 2011. *Recommender Systems: An Introduction*. Cambridge University Press, Cambridge, MA.

Jing Jiang, Jie Lu, Guangquan Zhang, and Guodong Long. 2011. Scaling-up item-based collaborative filtering recommendation algorithm based on Hadoop. In *Proceedings of the 2011 IEEE World Congress on Services (SERVICES'11)*. 490–497.

Christopher C. Johnson. 2014. Logistic matrix factorization for implicit feedback data. In *Proceedings of the NIPS 2014 Workshop on Distributed Matrix Computations*.

Bhargav Kanagal, Amr Ahmed, Sandeep Pandey, Vanja Josifovski, Jeff Yuan, and Lluis Garcia-Pueyo. 2012. Supercharging recommender systems using taxonomies for learning user purchase behavior. *Proceedings of the VLDB Endowment* 5, 10, 956–967. http://dl.acm.org/citation.cfm?id=2336664.2336669

Efthalia Karydi and Konstantinos Margaritis. 2012a. Multithreaded implementation of the slope one algorithm for collaborative filtering. In *Proceedings of 8th International Conference on Artificial Intelligence Applications and Innovations (AIAI'12)*.

Efthalia Karydi and Konstantinos Margaritis. 2012b. Parallel implementation of the slope one algorithm for collaborative filtering. In *Proceedings of the 16th Panhellenic Conference of Informatics (PCI'12)*.

Kimikazu Kato and Tikara Hosino. 2010a. Solving *k*-nearest neighbor problem on multiple graphics processors. In *Proceedings of the 2010 10th IEEE/ACM International Conference on Cluster, Cloud, and Grid Computing (CCGRID'10)*. IEEE, Los Alamitos, CA, 769–773. DOI: http://dx.doi.org/10.1109/CCGRID.2010.47

Kimikazu Kato and Tikara Hosino. 2010b. Singular value decomposition for collaborative filtering on a GPU. *IOP Conference Series: Materials Science and Engineering* 10, 1, 012017.

A. Kumar and P. Thambidurai. 2010. Collaborative Web recommendation systems—a survey approach. *Global Journal of Computer Science and Technology* 9, 5, 30–35.

Neeraj Kumar, Naveen Chilamkurti, and Jong-Hyouk Lee. 2012. Distributed context aware collaborative filtering approach for P2P service selection and recovery in wireless mesh networks. *Peer-to-Peer Networking and Applications* 5, 4, 350–362. DOI: http://dx.doi.org/10.1007/s12083-012-0156-4

Vipin Kumar, Ananth Grama, Anshul Gupta, and George Karypis. 1994. *Introduction to Parallel Computing: Design and Analysis of Algorithms*. Benjamin-Cummings Publishing, Redwood City, CA.

Bongjune Kwon and Hyuk Cho. 2010. Scalable co-clustering algorithms. In *Algorithms and Architectures for Parallel Processing*. Lecture Notes in Computer Science, Vol. 6081. Springer, 32–43. DOI: http://dx.doi.org/10.1007/978-3-642-13119-6_3

S. Lahabar and P. J. Narayanan. 2009. Singular value decomposition on GPU using CUDA. In *Proceedings of the IEEE International Parallel and Distributed Processing Symposium (IPDPS'09)*. 1–10. DOI: http://dx.doi.org/10.1109/IPDPS.2009.5161058

Joonseok Lee, Samy Bengio, Seungyeon Kim, Guy Lebanon, and Yoram Singer. 2014. Local collaborative ranking. In *Proceedings of the 23rd International Conference on World Wide Web (WWW'14)*. ACM, New York, NY, 85–96.

Joonseok Lee, Seungyeon Kim, Guy Lebanon, and Yoram Singer. 2013. Local low-rank matrix approximation. *Journal of Machine Learning* 28, 82–90.

Boduo Li, Sandeep Tata, and Yannis Sismanis. 2013. Sparkler: Supporting large-scale matrix factorization. In *Proceedings of the 16th International Conference on Extending Database Technology (EDBT'13)*. ACM, New York, NY, 625–636. DOI: http://dx.doi.org/10.1145/2452376.2452449

Mu Li, David G. Andersen, Jun Woo Park, Alexander J. Smola, Amr Ahmed, Vanja Josifovski, James Long, Eugene J. Shekita, and Bor-Yiing Su. 2014. Scaling distributed machine learning with the parameter server. In *Proceedings of the 11th USENIX Conference on Operating Systems Design and Implementation (OSDI'14)*. 583–598. http://dl.acm.org/citation.cfm?id=2685048.2685095

Ruifeng Li, Yin Zhang, Haihan Yu, Xiaojun Wang, Jiangqin Wu, and Baogang Wei. 2011. A social network-aware top-N recommender system using GPU. In *Proceedings of the 11th Annual International ACM/IEEE Joint Conference on Digital Libraries (JCDL'11)*. ACM, New York, NY, 287–296. DOI: http://dx.doi.org/10.1145/1998076.1998131

H. Liang, J. Hogan, and Y. Xu. 2010. Parallel user profiling based on folksonomy for large scaled recommender systems: An implementation of cascading MapReduce. In *Proceedings of the 2010 IEEE International Conference on Data Mining Workshops (ICDMW'10)*. 154–161.

Hamilton Link, Jared Saia, Randall Laviolette, and Terran Lane. 2005. Distributed Recommender Systems and the Network Topologies That Love Them. Available at https://www.semanticscholar.org/.

Zhaobin Liu, Wenyu Qu, Haitao Li, and Changsheng Xie. 2010. A hybrid collaborative filtering recommendation mechanism for P2P networks. *Future Generation Computer Systems* 26, 8, 1409–1417.

Zhiyuan Liu, Yuzhou Zhang, Edward Y. Chang, and Maosong Sun. 2011. PLDA+: Parallel latent Dirichlet allocation with data placement and pipeline processing. *ACM Transactions on Intelligent Systems and Technology* 2, 3, Article No. 26. DOI:http://dx.doi.org/10.1145/1961189.1961198

G. Louppe and P. Geurts. 2010. A zealous parallel gradient descent algorithm. In *Proceedings of the NIPS 2010 Workshop on Learning on Cores, Clusters, and Clouds*.

Yucheng Low, Joseph Gonzalez, Aapo Kyrola, Danny Bickson, Carlos Guestrin, and Joseph M. Hellerstein. 2010. GraphLab: A New Framework for Parallel Machine Learning. Available at http://graphlab.org/.

Jie Lu, Dianshuang Wu, Mingsong Mao, Wei Wang, and Guangquan Zhang. 2015. Recommender system application developments: A survey. *Decision Support Systems* 74, 12–32. DOI:http://dx.doi.org/10.1016/j.dss.2015.03.008

Linyuan Lü, Matús Medo, Chi Ho Yeung, Yi-Cheng Zhang, Zi-Ke Zhang, and Tao Zhou. 2012. Recommender systems. *Physics Reports* 519, 1, 1–49. DOI:http://dx.doi.org/10.1016/j.physrep.2012.02.006

Apache Mahout. 2016, What Is Apache Mahout? Retrieved July 18, 2016, from http://mahout.apache.org/.

Brian McFee, Thierry Bertin-Mahieux, Daniel P. W. Ellis, and Gert R. G. Lanckriet. 2012. The Million Song Dataset challenge. In *Proceedings of the 21st International Conference on World Wide Web (WWW'12 Companion)*. ACM, New York, NY, 909–916. DOI:http://dx.doi.org/10.1145/2187980.2188222

Metacritic. 1993. Home Page. Retrieved July 18, 2016, from http://www.metacritic.com.

Bradley N. Miller, Joseph A. Konstan, and John Riedl. 2004. PocketLens: Toward a personal recommender system. *ACM Transactions on Information Systems* 22, 3, 437–476.

Chandima Hewa Nadungodage, Yuni Xia, Jaehwan John Lee, Myungcheol Lee, and Choon Seo Park. 2013. GPU accelerated item-based collaborative filtering for big-data applications. In *Proceedings of the 2013 IEEE International Conference on Big Data.* 175–180. http://dx.doi.org/10.1109/BigData.2013.6691571

A. Narang, R. Gupta, A. Joshi, and V. K. Garg. 2010. Highly scalable parallel collaborative filtering algorithm. *Proceedings of the 2010 International Conference on High Performance Computing (HiPC'10).* 1–10.

Ankur Narang, Abhinav Srivastava, and Naga Praveen Kumar Katta. 2011a. High Performance Distributed Co-Clustering and Collaborative Filtering. Retrieved July 18, 2016, from http://domino.watson.ibm.com/library/Cyberdig.nsf/papers/E9F8290F6B662AEC8525795300452695.

Ankur Narang, Abhinav Srivastava, and Naga Praveen Kumar Katta. 2011b. Distributed scalable collaborative filtering algorithm. In *Proceedings of the 17th International Conference on Parallel Processing—Volume Part I.* 353–365.

Ankur Narang, Abhinav Srivastava, and Naga Praveen Kumar Katta. 2012. High performance offline and online distributed collaborative filtering. In *Proceedings of the 2012 IEEE 12th International Conference on Data Mining (ICDM'12)*. 549–558. DOI:http://dx.doi.org/10.1109/ICDM.2012.128

Netflix. 2009. Netflix Prize. Retrieved March 14, 2014, from http://www.netflixprize.com/.

Cyprien Noel and Simon Osindero. 2014. Dogwild!—Distributed Hogwild for CPU & GPU. Retrieved July 18, 2016, from http://stanford.edu/~rezab/nips2014workshop/submits/dogwild.pdf.

Martin Odersky, Philippe Altherr, Vincent Cremet, Burak Emir, Sebastian Maneth, Stephane Micheloud, Nikolay Mihaylov, Michel Schinz, Erik Stenman, and Matthias Zenger. 2004. *An Overview of the Scala Programming Language*. Technical Report. Ecole Polytechnique Federale de Lausanne, Lausanne, Switzerland.

Tomas Olsson. 1998. Decentralised social filtering based on trust. In *Proceedeings of the AAAI 1998 Recommender Systems Workshop*. 84–88.

Deuk Hee Park, Hyea Kyeong Kim, Il Young Choi, and Jae Kyeong Kim. 2012. A literature review and classification of recommender systems research. *Expert Systems with Applications* 39, 11, 10059–10072. DOI:http://dx.doi.org/10.1016/j.eswa.2012.02.038

Pittsburgh Supercomputing Center. 2009. Blacklight. Retrieved March 14, 2014, from http://www.psc.edu/.

Benjamin Recht and Christopher Re. 2011. Parallel stochastic gradient algorithms for large-scale matrix completion. Submitted for publication. http://pages.cs.wisc.edu/~brecht/publications.html.

K. N. Rao and V. G. Talwar. 2008. Application domain and functional classification of recommender systems—a survey. *DESIDOC Journal of Library and Information Technology* 28, 3, 17–35.

Benjamin Recht, Christopher Re, Stephen J. Wright, and Feng Niu. 2011. Hogwild: A lock-free approach to parallelizing stochastic gradient descent. In *Advances in Neural Information Processing Systems 24 (NIPS'11)*. 693–701.

Paul Resnick, Neophytos Iacovou, Mitesh Suchak, Peter Bergstrom, and John Riedl. 1994. *GroupLens: An Open Architecture for Collaborative Filtering of Netnews*. ACM, New York, NY. http://portal.acm.org/citation.cfm?id=192844.192905&type=series

Francesco Ricci, Lior Rokach, Bracha Shapira, and Paul B. Kantor (Eds.). 2011. *Recommender Systems Handbook*. Springer.

Matei Ripeanu, Ian Foster, and Adriana Iamnitchi. 2002. Mapping the Gnutella network: Properties of large-scale peer-to-peer systems and implications for system design. *IEEE Internet Computing Journal* 6, 1, 50–57.

Stephen E. Robertson, Evangelos Kanoulas, and Emine Yilmaz. 2010. Extending average precision to graded relevance judgments. In *Proceedings of the 33rd International ACM SIGIR Conference on Research and Development in Information Retrieval (SIGIR'10)*. ACM, New York, NY, 603–610.

Rotten Tomatoes. 1998. Home Page. Retrieved July 18, 2016, from http://www.rottentomatoes.com.

Giancarlo Ruffo and Rossano Schifanella. 2009. A peer-to-peer recommender system based on spontaneous affinities. *ACM Transactions on Internet Technology* 9, 1, Article No. 4. DOI:http://dx.doi.org/10.1145/1462159.1462163

Atisha Sachan and Vineet Richariya. 2013. A survey on recommender systems based on collaborative filtering technique. *International Journal of Innovations in Engineering and Technology* 2, 2, 1–7. http://ijiet.com/issues/volume-2-issue-2-april-2013/.

Ruslan Salakhutdinov, Andriy Mnih, and Geoffrey Hinton. 2007. Restricted Boltzmann machines for collaborative filtering. In *Proceedings of the 24th International Conference on Machine Learning (ICML'07)*. ACM, New York, NY, 791–798. DOI:http://dx.doi.org/10.1145/1273496.1273596

Badrul M. Sarwar, George Karypis, Joseph A. Konstan, and John T. Riedl. 2000a. Analysis of recommendation algorithms for e-commerce. In *Proceedings of the ACM E-Commerce 2000 Conference.*

Badrul M. Sarwar, George Karypis, Joseph A. Konstan, and John T. Riedl. 2000b. Application of dimensionality reduction in recommender system—a case study. *Architecture* 1625, 1, 2648. http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.29.8381&rep=rep1&type=pdf.

Sebastian Schelter, Christoph Boden, and Volker Markl. 2012. Scalable similarity-based neighborhood methods with MapReduce. In *Proceedings of the 6th ACM Conference on Recommender Systems (RecSys'12)*. ACM, New York, NY, 163–170. DOI:http://dx.doi.org/10.1145/2365952.2365984

Sebastian Schelter, Christoph Boden, Martin Schenck, Alexander Alexandrov, and Volker Markl. 2013. Distributed matrix factorization with MapReduce using a series of broadcast-joins. In *Proceedings of the 7th ACM Conference on Recommender Systems (RecSys'13)*. ACM, New York, NY, 281–284. DOI:http://dx.doi.org/10.1145/2507157.2507195

Sebastian Schelter, Venu Satuluri, and Reza Zadeh. 2014. Factorbird—a parameter server approach to distributed matrix factorization. arXiv:1411.0602.

Yue Shi, Alexandros Karatzoglou, Linas Baltrunas, Martha Larson, and Alan Hanjalic. 2013. GAPfm: Optimal top-N recommendations for graded relevance domains. In *Proceedings of the 22nd ACM International Conference on Conference on Information and Knowledge Management (CIKM'13)*. ACM, New York, NY, 2261–2266.

Yue Shi, Martha Larson, and Alan Hanjalic. 2014. Collaborative filtering beyond the user-item matrix: A survey of the state of the art and future challenges. *ACM Computing Surveys* 47, 1, Article No. 3. DOI:http://dx.doi.org/10.1145/2556270

Evan R. Sparks, Ameet Talwalkar, Virginia Smith, Jey Kottalam, Xinghao Pan, Joseph E. Gonzalez, Michael J. Franklin, Michael I. Jordan, and Tim Kraska. 2013. MLI: An API for distributed machine learning. arXiv:1310.5426. http://dblp.uni-trier.de/db/journals/corr/corr1310.html#SparksTSKPGFJK13.

Stratosphere. 2009. Home Page. Retrieved July 18, 2016, from http://stratosphere.eu/.

Xiaoyuan Su and Taghi M. Khoshgoftaar. 2009. A survey of collaborative filtering techniques. *Advances in Artificial Intelligence* 2009, Article No. 4.

Lei Tang and Patrick Harrington. 2013. Scaling matrix factorization for recommendation with randomness. In *Proceedings of the 22nd International Conference on World Wide Web Companion (WWW'13 Companion)*. 39–40. http://dl.acm.org/citation.cfm?id=2487788.2487801

Christina Teflioudi, Faraz Makari, and Rainer Gemulla. 2012. Distributed matrix completion. In *Proceedings of the IEEE International Conference on Data Mining (ICDM'12).*

P. B. Thorat, R. M. Goudar, and S. Barve. 2015. Survey on collaborative filtering, content-based filtering and hybrid recommendation system. *International Journal of Computer Applications* 110, 4, 31–36.

Dan-Cristian Tomozei and Laurent Massoulié. 2011. Distributed user profiling via spectral methods. arXiv:1109.3318.

A. Tripathy, S. Mohan, and R. Mahapatra. 2012. Optimizing a collaborative filtering recommender for many-core processors. In *Proceedings of the 2012 IEEE 6th International Conference on Semantic Computing (ICSC'12)*. 261–268. DOI:http://dx.doi.org/10.1109/ICSC.2012.58

Amund Tveit. 2001. Peer-to-peer based recommendations for mobile commerce. In *Proceedings of the 1st International Workshop on Mobile Commerce (WMC'01)*. ACM, New York, NY, 26–29. DOI:http://dx.doi.org/10.1145/381461.381466

Madeleine Udell, Corinne Horn, Reza Zadeh, and Stephen Boyd. 2014. Generalized low rank models. In *Proceedings of the NIPS 2014 Workshop on Distributed Matrix Computations*.

University of Waikato. 1997. Weka 3: Data Mining Software in Java. Retrieved July 18, 2016, from http://www.cs.waikato.ac.nz/ml/weka/.

Katrien Verbert, Nikos Manouselis, Xavier Ochoa, Martin Wolpers, Hendrik Drachsler, Ivana Bosnic, and Erik Duval. 2012. Context-aware recommender systems for learning: A survey and future challenges. *IEEE Transactions on Learning Technologies* 5, 4, 318–335. DOI:http://dx.doi.org/10.1109/TLT.2012.11

Bondhugula Vinay, Govindaraju Naga, and Manocha Dinesh. 2006. *Fast SVD on Graphics Processors*. Technical Report. University of North Carolina, Chapel Hill, NC. http://gamma.cs.unc.edu/NUMERIC/SVD/.

Emmanouil Vozalis and Konstantinos Margaritis. 2003. Analysis of recommender systems' algorithms. In *Proceedings of the 6th Hellenic European Conference on Computer Mathematics and Its Applications (HERCMA'03)*. 732–745.

Jun Wang, Johan Pouwelse, Reginald L. Lagendijk, and Marcel J. T. Reinders. 2006. Distributed collaborative filtering for peer-to-peer file sharing systems. In *Proceedings of the 2006 ACM Symposium on Applied Computing (SAC'06)*. ACM, New York, NY, 1026–1030. DOI:http://dx.doi.org/10.1145/1141277.1141522

Yi Wang, Hongjie Bai, Matt Stanton, Wen-Yen Chen, and Edward Y. Chang. 2009. PLDA: Parallel latent Dirichlet allocation for large-scale applications. In *Proceedings of the 5th International Conference on Algorithmic Aspects in Information and Management (AAIM'09)*. 301–314. DOI:http://dx.doi.org/10.1007/978-3-642-02158-9_26

Zhongya Wang, Ying Liu, and Pengshan Ma. 2014. A CUDA-enabled parallel implementation of collaborative filtering. *Procedia Computer Science* 30, 66–74.

Yao Wu, Qiang Yan, Danny Bickson, Yucheng Low, and Qing Yang. 2011. Efficient multicore collaborative filtering. arXiv:1108.2580

Bo Xie, Peng Han, Fan Yang, Rui-Min Shen, Hua-Jun Zeng, and Zheng Chen. 2007. DCFLA: A distributed collaborative-filtering neighbor-locating algorithm. *Information Sciences* 177, 6, 1349–1363. DOI:http://dx.doi.org/10.1016/j.ins.2006.09.005

Ruzhi Xu, Shuaiqiang Wang, Xuwei Zheng, and Yinong Chen. 2014. Distributed collaborative filtering with singular ratings for large scale recommendation. *Journal of Systems and Software* 95, 231–241.

Xiwang Yang, Yang Guo, Yong Liu, and Harald Steck. 2014. A survey of collaborative filtering based social recommender systems. *Computer Communications* 41, 1–10. DOI:http://dx.doi.org/10.1016/j.comcom.2013.06.009

Hsiang-Fu Yu, Cho-Jui Hsieh, Si Si, and Inderjit S. Dhillon. 2012. Scalable coordinate descent approaches to parallel matrix factorization for recommender systems. In *Proceedings of the IEEE International Conference on Data Mining (ICDM'12)*. 765–774.

Zhi-Qin Yu, Xing-Jian Shi, Ling Yan, and Wu-Jun Li. 2014. Distributed stochastic ADMM for matrix factorization. In *Proceedings of the 23rd ACM International Conference on Information and Knowledge Management (CIKM'14)*. ACM, New York, NY, 1259–1268. DOI:http://dx.doi.org/10.1145/2661829.2661996

H. Yun, P. Raman, and S. Vishwanathan. 2014. Ranking via robust binary classification. In *Advances in Neural Information Processing Systems 27 (NIPS'14)*. 2582–2590. http://papers.nips.cc/paper/5363-ranking-via-robust-binary-classification.pdf.

Matei Zaharia, Mosharaf Chowdhury, Tathagata Das, Ankur Dave, Justin Ma, Murphy McCauley, Michael J. Franklin, Scott Shenker, and Ion Stoica. 2012. Resilient distributed datasets: A fault-tolerant abstraction for in-memory cluster computing. In *Proceedings of the 9th USENIX Conference on Networked Systems Design and Implementation (NSDI'12)*. 2. http://dl.acm.org/citation.cfm?id=2228298.2228301

David Zastrau and Stefan Edelkamp. 2012. Stochastic gradient descent with GPGPU. In *Proceedings of the 35th Annual German Conference on Advances in Artificial Intelligence (KI'12)*. 193–204. DOI:http://dx.doi.org/10.1007/978-3-642-33347-7_17

Gao Zhanchun and Liang Yuying. 2012. Improving the collaborative filtering recommender system by using GPU. In *Proceedings of the 2012 International Conference on Cyber-Enabled Distributed Computing and Knowledge Discovery (CyberC'12)*. 330–333. DOI:http://dx.doi.org/10.1109/CyberC.2012.62

Yongfeng Zhang, Min Zhang, Yiqun Liu, Shaoping Ma, and Shi Feng. 2013. Localized matrix factorization for recommendation based on matrix block diagonal forms. In *Proceedings of the 22nd International Conference on World Wide Web (WWW'13)*. 1511–1520.

Zi-Ke Zhang, Tao Zhou, and Yi-Cheng Zhang. 2011. Tag-aware recommender systems: A state-of-the-art survey. *Journal of Computer Science and Technology* 26, 5, 767–777.

Jerry Zhao and Jelena Pjesivac-Grbovic. 2009. MapReduce: The Programming Model and Practice. Retrieved July 18, 2016, from http://research.google.com/archive/papers/mapreduce-sigmetrics09-tutorial.pdf.

Zhi-Dan Zhao and Ming-Sheng Shang. 2010. User-based collaborative-filtering recommendation algorithms on Hadoop. In *Proceedings of the 2010 3rd International Conference on Knowledge Discovery and Data Mining (KDDM'10)*. 478–481.

Yunhong Zhou, Dennis Wilkinson, Robert Schreiber, and Rong Pan. 2008. Large-scale parallel collaborative filtering for the Netflix Prize. *Algorithmic Aspects in Information and Management* 337–348.

Yong Zhuang, Wei-Sheng Chin, Yu-Chin Juan, and Chih-Jen Lin. 2013. A fast parallel SGD for matrix factorization in shared memory systems. In *Proceedings of the 7th ACM Conference on Recommender Systems (RecSys'13)*. ACM, New York, NY, 249–256. DOI: http://dx.doi.org/10.1145/2507157.2507164

Martin Zinkevich, Markus Weimer, Alex Smola, and Lihong Li. 2010. Parallelized stochastic gradient descent. In *Advances in Neural Information Processing Systems 23 (NIPS'10)*. 2595–2603.

Received April 2014; revised May 2016; accepted June 2016