Collaborative Filtering for Orkut Communities: Discovery of User Latent Behavior

Wen-Yen Chen*
University of California
Santa Barbara, CA 93106
wychen@cs.ucsb.edu

Jon-Chyuan Chu*
MIT
Cambridge, MA 02139
jonchu88@mit.edu

Junyi Luan*
Peking University
Beijing 100084, China
luanjunyi@gmail.com

Hongjie Bai
Google Research
Beijing 100084, China
hjbai@google.com

Yi Wang
Google Research
Beijing 100084, China
wyi@google.com

Edward Y. Chang
Google Research
Beijing 100084, China
edchang@google.com

ABSTRACT

Users of social networking services can connect with each other by forming communities for online interaction. Yet as the number of communities hosted by such websites grows over time, users have even greater need for effective community recommendations in order to meet more users. In this paper, we investigate two algorithms from very different domains and evaluate their effectiveness for personalized community recommendation. First is association rule mining (ARM), which discovers associations between sets of communities that are shared across many users. Second is latent Dirichlet allocation (LDA), which models user-community co-occurrences using latent aspects. In comparing LDA with ARM, we are interested in discovering whether modeling low-rank latent structure is more effective for recommendations than directly mining rules from the observed data. We experiment on an Orkut data set consisting of $492,104$ users and $118,002$ communities. Our empirical comparisons using the top-k recommendations metric show that LDA performs consistently better than ARM for the community recommendation task when recommending a list of 4 or more communities. However, for recommendation lists of up to 3 communities, ARM is still a bit better. We analyze examples of the latent information learned by LDA to explain this finding. To efficiently handle the large-scale data set, we parallelize LDA on distributed computers [1] and demonstrate our parallel implementation's scalability with varying numbers of machines.

Categories and Subject Descriptors

H.2.8 [**Database Management**]: Database Applications—*Data Mining*; H.3.3 [**Information Storage and Retrieval**]: Information Search and Retrieval—*Information Filtering*; H.3.4 [**Information Storage and Retrieval**]: Systems and Software—*Distributed Systems*

*Work done while the first three authors were interns at Google Research, Beijing.

General Terms

Algorithms, Experimentation

Keywords

Recommender systems, collaborative filtering, association rule mining, latent topic models, data mining

1. INTRODUCTION

Much information is readily accessible online, yet we as humans have no means of processing all of it. To help users overcome the information overload problem and sift through huge amounts of information efficiently, *recommender systems* have been developed to generate suggestions based on user preferences. In general, recommender systems can be classified into two categories: *content-based filtering* and *collaborative filtering*. Content-based filtering analyzes the association between user profiles and the descriptions of items. To recommend new items to a user, the content-based filtering approach matches the new items' descriptions to those items known to be of interest to the user. On the other hand, the collaborative filtering (CF) approach does not need content information to make recommendations. The operative assumption underlying CF is that users who had similar preferences in the past are likely to have similar preferences in the future. As such, CF uses information about similar users' behaviors to make recommendations. Besides avoiding the need for collecting extensive information about items and users, CF requires no domain knowledge and can be easily adopted across different recommender systems.

In this paper, we focus on applying CF to community recommendation. To investigate which notions of similarity are most useful for this task, we examine two approaches from different fields. First, association rule mining (ARM) [2] is a data mining algorithm that finds association rules based on frequently co-occurring sets of communities and then makes recommendations based on the rules. For example, if users in the community "New York Yankees" usually join the community "MLB" (for Major League Baseball), then ARM will discover a rule connecting communities "New York Yankees" and "MLB." If you join "New York Yankees," you can be recommended "MLB." Generally speaking, ARM can discover explicit relations between communities based on their co-occurrences across multiple users.

Second, latent Dirichlet allocation (LDA) [5] is a machine-learning algorithm that models user-community co-occurrences using latent aspects and makes recommendations based on the learned model parameters. Unlike ARM, LDA models the implicit relations between communities through the set of latent aspects. Following the previous example, suppose we have another community called "New York Mets" that shares many users in common with "MLB" but not with "New York Yankees." If you are a user joining "New York Mets," you would not receive recommendations from ARM for "New York Yankees," because there are not enough "New York Mets"-"New York Yankees" co-occurrences to support such a rule. This is the problem of *sparseness* in explicit co-occurrence. On the other hand, LDA can possibly recommend "New York Yankees" because of there may exist an implicit relation (semantically, "baseball") among the three communities. However, implicit co-occurrence is not always inferred correctly. Suppose the community "Boston Red Sox" occurs frequently with "MLB" but not "New York Yankees." With LDA, if you were to join "Boston Red Sox," you might receive a recommendation for "New York Yankees," which is a bad recommendation as the two teams have a longstanding rivalry. This is the problem of *noise* in inferred implicit co-occurrence.

Explicit relations suffer from the problem of sparseness, but are usually quite accurate. Modeling implicit relations can overcome the sparseness problem; however, doing so may introduce noise. In what scenarios does each type of relation manifest its strengths and weaknesses? In particular, how do membership size (the number of communities joined by a given user) and community size (the number of users in a given community) affect sparseness of explicit relations mined using ARM?

- **Membership size**: Would explicit relations be more effective at recommendations for *active users*, ones who have joined many communities?
- **Community size**: Would implicit relations be more effective at recommending *new or niche communities* with few members?

Similarly, how do membership spread (the diversity among the communities that a user joins) and community spread (the diversity among the users of a particular community) affect noise in the implicit relations inferred by a latent topic model such as LDA?

- **Membership spread**: Would explicit relations be more effective at recommendations for a *diverse user*, one who is involved in a miscellaneous set of communities?
- **Community spread**: Would implicit relations be more effective at recommending *umbrella communities*, those composed of many smaller, tighter sub-communities or many non-interacting members?

It remains unclear how sparseness and noise come into play in collaborative filtering. Both explicit and implicit relations are arguably important for community recommendations to overcome the problems of sparseness and noise, but how can they be maximally leveraged for recommendations? Up until now, little has been done in the way of performance comparisons. This paper makes the following three contributions:

- We apply both algorithms to an Orkut data set consisting of $492,104$ users and $118,002$ communities. Our empirical comparisons using the top-k recommendations metric show a surprisingly intuitive finding: that LDA performs consistently better than ARM for the community recommendation task when recommending a list of 4 or more communities. However, for recommending up to 3 communities, ARM is still a bit better.
- We further analyze the latent information learned by LDA to help explain this finding.
- We parallelize LDA to take advantage of the distributed computing infrastructure of modern data centers. Our scalability study on the same Orkut data set shows that our parallelization can reduce the training time from 8 hours to less than 46 minutes using up to 32 machines (We have made parallel LDA *open-source* [1]).

The remainder of this paper is organized as follows: In Section 2, we briefly introduce ARM and LDA and works related to them. In Section 3, we show how ARM and LDA can be adapted for the community recommendation task. We present our parallelization framework of LDA in Section 4 and an empirical study on our Orkut data set in Section 5. Finally, we offer our concluding remarks in Section 6.

2. RELATED WORK

Association rule mining (ARM) is a well researched algorithm for discovering of correlations between sets of items. It was originally motivated by analysis of supermarket transaction data to discover which itemsets are often purchased together. For example, the rule $X \Rightarrow Y$ means customers who purchased X would likely also purchase Y. To select interesting rules from the set of all possible rules, *support* and *confidence* are used as constraints. The *support* of an itemset X, $\text{supp}(X)$, is defined as the number of transactions in the data set that contain the itemset. A *frequent itemset* is one whose support value exceeds a predefined threshold. An association rule is of the form $X \Rightarrow Y$, where X and Y are itemsets. The *support* of a rule is defined as $\text{supp}(X \Rightarrow Y) = \text{supp}(X \cup Y)$, and the *confidence* of a rule is defined as $\text{conf}(X \Rightarrow Y) = \text{supp}(X \cup Y)/\text{supp}(X)$. Association rule mining is the process of finding all rules whose support and confidence exceed specified thresholds.

Since ARM was first introduced in [2], various algorithms have been designed to mine frequent itemsets efficiently. In [3] and [18], a fast algorithm called Apriori was proposed to exploit the monotonicity property of the support of itemsets and the confidence of association rules. Subsequently, FP-growth [9] employed depth-first search for mining frequent itemsets and has been shown to outperform Apriori in most cases. ARM has been applied to several CF tasks. The work of [15] proposed to use ARM for mining user access patterns and predicted Web pages requests. In [14], authors examined the robustness of a recommendation algorithm based on ARM.

Latent Dirichlet allocation (LDA) [5], as used in document modeling, assumes a generative probabilistic model in which documents are represented as random mixtures over latent topics, where each topic is characterized by a probability distribution over words. In LDA, the generative process

Table 1: Notations. The following notations are used in the paper.

N	number of users
M	number of communities
K	number of topics
P	number of machines
$u_1, \ldots, u_N \in U$	a set of users
$c_1, \ldots, c_M \in C$	a set of communities
$z_1, \ldots, z_K \in Z$	a set of topics

Table 2: Each user is a transaction and his joined communities are items.

User	Communities
u_1	$\{c_1, c_3, c_7\}$
u_2	$\{c_3, c_7, c_8, c_9\}$
u_3	$\{c_2, c_3, c_8\}$
u_4	$\{c_1, c_8, c_9\}$

(a) Frequent itemsets and their support. The minimum support threshold is 2.

Frequent Itemsets	Support
$\{c_1\}$	2
$\{c_3\}$	3
$\{c_7\}$	2
$\{c_8\}$	3
$\{c_9\}$	2
$\{c_3, c_7\}$	2
$\{c_3, c_8\}$	2
$\{c_8, c_9\}$	2

(b) Association rules with their support and confidence values.

Association Rules	Support	Confidence
$c_3 \Rightarrow c_7$	2	66.7%
$c_3 \Rightarrow c_8$	2	66.7%
$c_7 \Rightarrow c_3$	2	100%
$c_8 \Rightarrow c_3$	2	66.7%
$c_8 \Rightarrow c_9$	2	66.7%
$c_9 \Rightarrow c_8$	2	100%

Table 3: Frequent itemsets and association rules.

consists of three steps. First, for each document, a multinomial distribution over topics is sampled from a Dirichlet prior. Second, each word in the document is assigned a single topic according to this distribution. Third, each word is generated from a multinomial distribution specific to the topic.

In [5], the parameters of LDA are estimated using an approximation technique called variational EM, since standard estimation methods are intractable. The work of [7] shows how Gibbs sampling, a simple and widely applicable Markov Chain Monte Carlo technique, could be applied to estimate parameters of LDA. By extending LDA, several other algorithms have been proposed to model publication (the Author-Topic model) [19] and email data (the Author-Recipient-Topic model) [13]. Not limited to text analysis, LDA has also been used to annotate images, where image segmentations (blobs) can be considered as words in an image document [4].

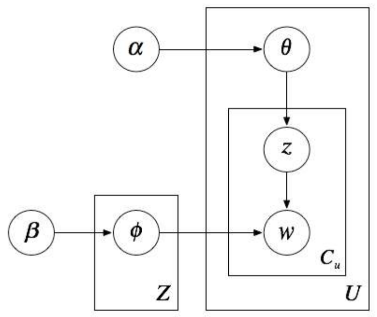


Figure 1: LDA model for user-community data.

3. RECOMMENDATION ALGORITHMS

To assist readers, Table 1 defines terms and notations used throughout this paper.

3.1 Association Rule Mining and Prediction

In ARM, we view each user as a transaction and his joined communities as items, as shown in Table 2. We employ the FP-growth algorithm [9] for mining frequent itemsets and use the discovered frequent itemsets to generate association rules. We generate first-order association rules for recommendations. That is, each item is iterated one by one as the precondition of the rule. For example, assuming the support threshold is 2, we can mine the frequent itemsets and their corresponding association rules as shown in Table 3. With the rules at hand, we can recommend communities to a user based on his joined communities. Specifically, we match each joined community with the precondition of the rules to recommend communities. We weight the recommended communities by summing up each corresponding rule's confidence. For example, suppose a user joins communities $\{c_7, c_8\}$. According to the association rules in Table 3 (b), three rules will be taken into consideration: $\{c_7 \Rightarrow c_3\}$, $\{c_8 \Rightarrow c_3\}$ and $\{c_8 \Rightarrow c_9\}$. In the end, the user will be recommended community c_3 with score 1.667 (1+0.667) and community c_9 with score 0.667.

3.2 LDA Training and Inference

In LDA, user-community data is entered as a membership count where the value is 1 (join) or 0 (not join). To train LDA, we then view the values as co-occurrence counts. Below we show how to estimate parameters using Gibbs sampling, then infer the community recommendation from the model parameters and analyze the computational complexity. As shown in Figure 1, we denote the per-user topic distribution as θ, each being drawn independently from a symmetric Dirichlet prior α, and the per-topic community distribution as ϕ, each being drawn from a symmetric Dirichlet prior β. For each occurrence, the topic assignment is sampled from:

$$P(z_i = j | w_i = c, \mathbf{z}_{-i}, \mathbf{w}_{-i}) \propto \frac{C^{CZ}_{cj} + \beta}{\sum_{c'} C^{CZ}_{c'j} + M\beta} \frac{C^{UZ}_{uj} + \alpha}{\sum_{j'} C^{UZ}_{uj'} + K\alpha}, \quad (1)$$

Table 4: Sample MPI functions.

MPI_Send:	Blocking send message to destination process.
MPI_Receive:	Blocking receive a message from source process.
MPI_Broadcast:	Broadcasts information to all processes.
MPI_AllGather:	Gathers the data contributed by each process on all processes.
MPI_Reduce:	Performs a global reduction (*i.e.*, sum) and returns the result to the specified root.
MPI_AllReduce:	Performs a global reduction (*i.e.*, sum) and returns the result on all processes.

where $z_i = j$ represents the assignment of the i_{th} community occurrence to topic j, $w_i = c$ represents the observation that the i_{th} community occurrence is the community c in the community corpus, $\mathbf{z}_{-i}$ represents all topic assignments not including the ith community occurrence, and $\mathbf{w}_{-i}$ represents all community occurrences not including the ith community occurrence. Furthermore, C^{CZ}_{cj} is the number of times community c is assigned to topic j, not including the current instance, and C^{UZ}_{uj} is the number of times topic j is assigned to user u, not including the current instance. From these count matrices, we can estimate the topic-community distributions ϕ and user-topic distribution θ by:

$$\phi_{cj} = \frac{C^{CZ}_{cj} + \beta}{\sum_{c'} C^{CZ}_{c'j} + M\beta},$$
$$\theta_{uj} = \frac{C^{UZ}_{uj} + \alpha}{\sum_{j'} C^{UZ}_{uj'} + K\alpha}, \quad (2)$$

where ϕ_{cj} is the probability of containing community c in topic j, and θ_{uj} is the probability of user u using topic j. The algorithm randomly assigns a topic to each community occurrence, updates the topic to each occurrence using Gibbs sampling (Equation 1), and then repeats the Gibbs sampling process to update topic assignments for several iterations.

Once we have learned the model parameters, we can infer user-community relations using Bayes' rule. For example, communities can be ranked for a given user according to the score ξ:

$$\xi_{cu} = \sum_z \phi_{cz}\theta_{uz}. \quad (3)$$

Communities with high scores but not joined by the user are good candidates for recommendation.

In each Gibbs sampling iteration, one needs to compute the posterior probability for each community occurrence:

$$P(z_i = j | w_i = c, \mathbf{z}_{-i}, \mathbf{w}_{-i}). \quad (4)$$

We assume that on average there are L community occurrences for every user and each $P(z_i = j|w_i = c, \mathbf{z}_{-i}, \mathbf{w}_{-i})$ consists of K topics. Then, the total computational complexity of running l Gibbs sampling iterations for N users is $O(K \cdot N \cdot L \cdot l)$.

4. PARALLELIZATION

We parallelized LDA with libraries of MPI [16] and MapReduce [6]. (For our parallel ARM effort, please consult [12].) With MPI, a program is loaded into the local memory of each machine, where every local process receives a unique ID. When needed, the processes can communicate and synchronize with each other by calling the MPI library functions shown in Table 4. With MapReduce, a user specifies Map and Reduce functions and the program reads and writes results to disks. For Gibbs sampling, which requires computation of multiple iterations, we face a tradeoff decision between efficiency (without disk IOs of MPI) and reliability (with disk IOs of MapReduce). For our empirical studies where the computation time is relatively short compared to mean-time-between-failures (MTBF) of machines, we chose the MPI implementation. (Though MPI itself does not perform logging for recovery, an application can perform checkpoints to achieve fault tolerance, which is an issue beyond the scope of this paper.) Since standard MPI implementations such as MPICH2[1] [8] cannot be directly ported to our distributed data centers, we implemented our own MPI system by modifying MPICH2. We have made LDA/MPI *open-source*; please visit [1] for details.

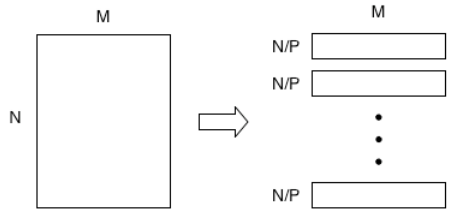


Figure 2: The user-community matrix is distributedly computed and stored on multiple machines.

The parameter estimation using Gibbs sampling in LDA can be divided into parallel subtasks. Suppose P machines are allocated in a distributed environment. Figure 2 shows that we construct N/P rows of the user-community matrix at each machine. Thus, each machine i only deals with a specified subset of users $U_i \subset U$, and is aware of all communities c. Recall that in LDA, community memberships are viewed as occurrence counts. Each machine i randomly initializes a topic assignment for every community occurrence and calculates local counts $C^{CZ}_{cj}(i)$ and $C^{U_iZ}_{uj}$. All local counts are then *reduces* (*i.e.*, sum) to the specified root, and root *broadcasts* the global count C^{CZ}_{cj} to all machines. Most communication occurs here, and this is a *MPI_AllReduce* function in MPI shown in Table 4. We perform Gibbs sampling simultaneously on each machine independently, and updates the topic assignment for each community occurrence. The sampling process is repeated for l iterations. We summarize the procedure in Algorithm 1.

The computational complexity for parallel LDA reduces to $O((K \cdot N \cdot L \cdot l)/P)$. As for communication complexity, C^{CZ}_{cj} is reduced and broadcasted among P machines in each training iteration through an *MPI_AllReduce* call. After some experiments, we use the *recursive-halving* and *recursive doubling* algorithms [20] for an efficient implementation of *MPI_AllReduce* functionality. Under this scheme,

[1]http://www.mcs.anl.gov/research/projects/mpich2

Algorithm 1: Parallel LDA

```
   Input: user-community matrix ($N \times M$)
   Output: $C^{CZ}_{cj}$, $C^{UZ}_{uj}$
   Variables:
   $l$: number of iterations
1  Each machine $i$ loads $N/P$ rows of input matrix and
   randomly assigns a topic to each community occurrence
2  for $iter = 1$ to $l$ do
3      Each machine $i$ calculates local counts
4          $C^{CZ}_{cj}(i)$ and $C^{U_iZ}_{uj}$
5      All local counts $C^{CZ}_{cj}(i)$ are reduced to the root
6          $C^{CZ}_{cj} = \sum_i C^{CZ}_{cj}(i)$
7      Root broadcasts global count $C^{CZ}_{cj}$ to all machines
8      foreach $u$ in $U_i$ do
9          foreach community occurrence do
10             Performs Gibbs sampling using Equation (1)
11             Updates the topic assignment
12         end
13     end
14 end
```

the communication cost becomes $O(\alpha \cdot \log(P) + \beta \cdot \frac{P-1}{P} KM + \gamma \cdot \frac{P-1}{P} KM)$ per iteration, where α is the startup time of a transfer, β is the transfer time per byte, and γ is the computation time per byte for performing the reduction operation locally on any machine.

5. EXPERIMENTS

We divide our experiments into two parts. The first part is conducted on an Orkut community data set to evaluate the recommendation quality of LDA and ARM using top-k recommendations metric. The second part is conducted on the same Orkut data set to investigate the scalability of our parallel implementation.

5.1 Recommendation Quality

5.1.1 The Orkut Data Set

According to Alexa web information service, Orkut is an extremely active social network service with more than two billion page views a day worldwide. Our community membership information data set was a filtered collection of Orkut in July 2007. To safeguard user privacy, all user and community data were anonymized as performed in [17]. After restricting our data to English communities only, we collected 492, 104 users and 118, 002 communities. The density of the user-community matrix (percentage of non-zero entries) is 0.01286%; the matrix is extremely sparse.

5.1.2 Evaluation Metric and Protocol

The metrics for evaluating recommendation algorithms can be divided into two classes: (1) Prediction accuracy metrics measure the difference between the true values and the predicted values. Commonly used metrics include Mean Absolute Error (MAE) and Root Mean Square Error (RMSE); (2) Ranking accuracy metrics measure the ability to produce an ordered list of items that matches how a user would have ordered the same items. These include the top-k recommendations [11] and Normalized Discounted Cumulative Gain (NDCG) [10]. Our overall goal is to measure the effectiveness of suggesting top-ranked items to a user. Besides, the predicted scores of recommended communities by LDA and ARM are not in same range. Thus, to fairly compare their performance, we employ the top-k recommendations metric. That is, each ranking algorithm needs to suggest the top k items to a user. We describe the evaluation protocol below.

First, for each user u, we randomly withhold one joined community c from his *original set* of joined communities to form user u's *training set*. The training sets for all users form the training input for both algorithms. Second, for each user u, we select $k-1$ additional random communities that were not in user u's original set; the withheld community c together with these $k-1$ other communities form user u's *evaluation set* (of size k).

For user u, LDA calculates the score for each of the k communities in the evaluation set using Equation 3. ARM assigns the score of each community based on the mined rules. Note that LDA is guaranteed to produce a well-defined score for each evaluation community. However, this is not the case for ARM. For a given user u, it is possible that an evaluation community will not be recommended back by ARM's rules, because no rules for it apply to user u (based on his training set). In this case, we assign such a "ruleless" community an infinite negative score to be worst ranked. Last, for each user u, we order the k communities in his evaluation set by their predicted score to obtain a corresponding rank between 1 and k for each.

Our objective is to find the relative placement of each user u's withheld community c. There are k possible ranks for c, ranging from the best rank where no random community precedes c in the ordered list, to worst rank where all of the random communities appear before c. The best result we can hope for is that community c will precede the $k-1$ random communities in our ranking.

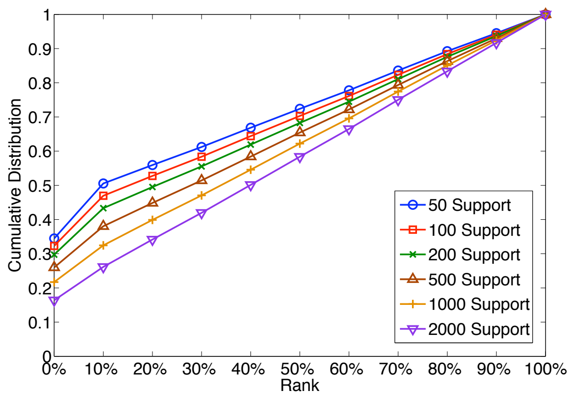
5.1.3 Results

We first describe the parameter settings, and then present the experimental results. For ARM, we ran a grid search over different support values (50, 100, 200, 500, 1000, and 2000); for LDA, we searched a grid over the number of topics (30, 60, 90, 120, and 150) and a grid of number of iterations (100, 200, 300, 400, and 500), respectively. The default value of α is $50/K$ (K is the number of topics), and the default value of β is 0.1. We set k to be 1001, so that the number of random communities selected for ranking evaluation is 1000. Overall, there are 492, 104 communities withheld from Orkut data set (one community withheld for each user).

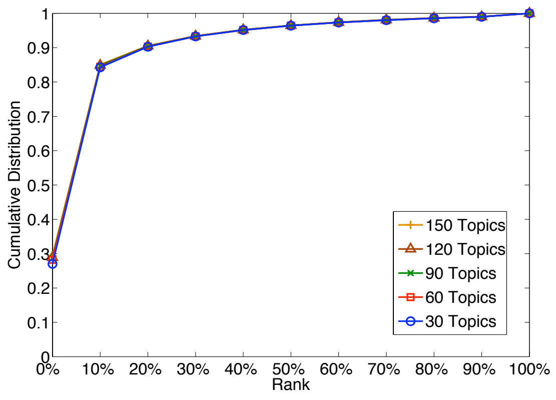
We present the experimental results in three perspectives: 1) top-k recommendation performance of applying LDA and ARM to the Orkut data set, 2) rank differences between LDA and ARM, and 3) the analysis of latent information learned from LDA.

Top-k recommendation performance. Figure 3 shows the cumulative distributions of ranks for withheld communities. For example, the 0% rank indicates that the withheld community is ranked 1 in the ordered list, while 100% indicates that it is ranked last in the list. The lower the rank, the more successful the recommendation. From the figure we can make the following three observations:

- ARM achieves the best performance when the support value is 50. When the support value is higher, the

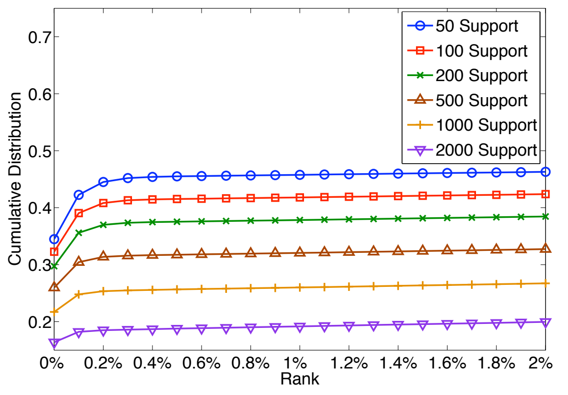


(a) ARM: macro view of top-k performance

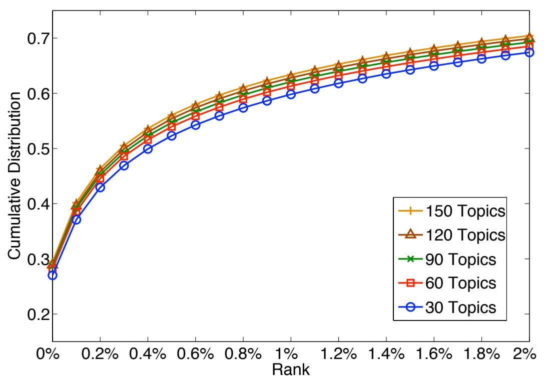


(b) LDA: macro view of top-k performance

Figure 3: The macro view ($0\% - 100\%$) of top-k recommendations performance: we plot the cumulative distributions of ranks for withheld communities in the ordered list. Here, the length of the ordered list is 1001. The x-axis is the percentile rank for the withheld community; the lower the rank, the more successful the recommendation.



(a) ARM: micro view of top-k performance



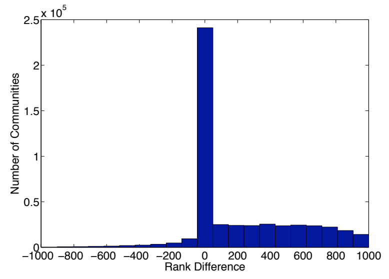
(b) LDA: micro view of top-k performance

Figure 4: The macro view (top 2%) of top-k recommendations performance. We plot the cumulative distributions of ranks for withheld communities in the ordered list. The x-axis is the percentile rank for the withheld community; the lower the rank, the more successful the recommendation.

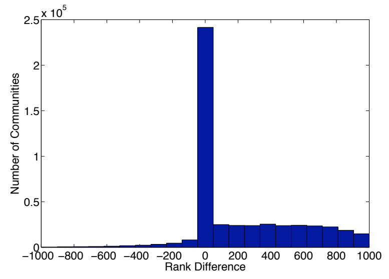
recommendation performance deteriorates. This is because when the support value is higher, there are fewer frequent itemsets being mined, and thus fewer rules are generated. Thus, the coverage of communities available for recommendations becomes smaller. Note that a support value of 50, at 0.01% of the number of transactions, is already extremely low in the ARM domain. Rules discovered via ARM with support values lower than this would likely not be statistically significant.

- LDA achieves more consistent performance than ARM. The performance gap of LDA between different number of topics are very minor and thus the performance curves basically overlap.
- ARM (with support values 50, 100, and 200) performs slightly better than LDA for the top 0% rank (the withheld community was ranked 1 in the ordered list of length 1001) and then LDA outperforms ARM for the rest of the ranks. Since recommender systems focus more on recommending top-ranked items to users, we zoom in on the top 2% performance for both LDA and ARM in Figure 4. We then can make two more findings. First, ARM (with support value 50) performs slightly better than LDA only before the top 0.4% rank. After that, LDA's performance curves climb more quickly than ARM's. Second, the ARM curves do not grow much between 0.2% ranks and 2% ranks. That is, for the top 2% ranks, ARM leans toward either assigning the very top rank, or the rank after 20, to the withheld communities.

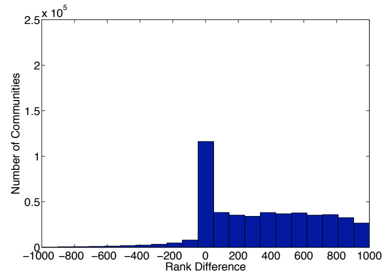
These findings lead to two interesting questions: 1) for a withheld community, what is the rank difference between LDA and ARM, and how do their parameters affect the differences? and 2) for what types of communities does LDA rank better or worse than ARM, and what are the character-



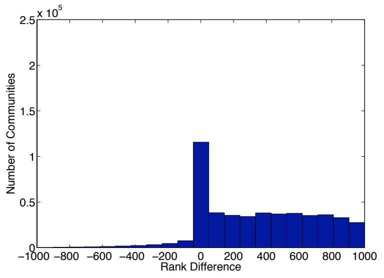
(a) ARM-50 vs. LDA-30



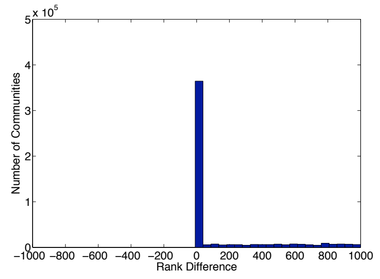
(b) ARM-50 vs. LDA-150



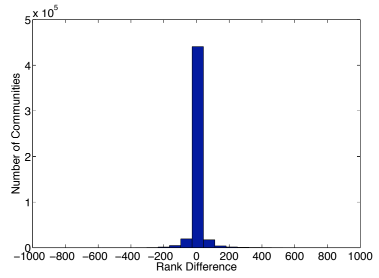
(c) ARM-2000 vs. LDA-30



(d) ARM-2000 vs. LDA-150



(e) ARM-2000 vs. ARM-50



(f) LDA-30 vs. LDA-150

Figure 5: Histograms of each withheld community's rank difference by various pairs of two algorithms. (a) ARM with support value 50 versus LDA with 30 topics; (b) ARM with support value 50 versus LDA with 150 topics; (c) ARM with support value 2000 versus LDA with 30 topics; (d) ARM with support value 2000 versus LDA with 150 topics; (e) ARM with support value 2000 versus support value 50; (f) LDA with 30 topics versus 150 topics. For each withheld community, the rank difference is calculated by subtracting the rank by the right-hand algorithm from the rank by the left-hand algorithm.

istics of these communities and their corresponding users? How does the latent information learned from LDA influence the ranking performance? We address these questions next.

Rank differences between LDA and ARM. We examine each withheld community's rank difference to plot the histograms shown in Figure 5. To calculate the rank difference, taking Figure 5 (a) for example, we subtract the rank by LDA-30-topics (the algorithm on the right-hand side) from the rank by ARM-50-support (the algorithm on the left-hand side). We obtained the remaining subfigures in a similar fashion.

We compare the best-performing ARM to the worst and best LDA in Figure 5 (a) and (b), respectively. Most of the withheld communities have positive rank difference rather than negative rank difference. This means LDA ranked most of the communities better than ARM did. Moreover, when LDA is better than ARM, it is likely that LDA is much better than ARM (high variance for positive rank differences). On the other hand, when ARM is better than LDA, it is likely to be only a little better than LDA (low variance for negative rank differences). Besides, when LDA trains with more topics, the rank differences from ARM do not differ much. This is expected, as LDA performs consistently with different numbers of topics, as shown in Figure 3.

We compare the worst-performing ARM to the worst and best LDA in Figure 5 (c) and (d), respectively. Comparing to Figure 5 (a) and (b), we observe a similar pattern, but fewer communities have rank difference zero. Instead, more communities have positive rank differences. This is due to ARM's higher support value – ARM has fewer association rules that would include the withheld community in the recommendation. Thus the withheld community is very likely to be assigned an infinite negative score and ranked worst.

Comparing the worst and best ARM in Figure 5 (e), we observe that there are far fewer communities with negative rank differences than positive rank differences. This again shows that ARM with a lower support value ranks better for the withheld communities, because it has more rules. Comparing the worst and best LDA in Figure 5 (f), we observe that unlike ARM, there is a nearly equal number of communities for positive and negative rank differences. Thus the performances of LDA with varying numbers of topics are similar.

Analysis of latent information learned from LDA. To analyze the types of communities for which LDA ranks better or worse than ARM, we investigate the latent information learned from LDA. In Figure 6, we show the topic distributions of each community in a user's training set on the left, and the topic distributions for the corresponding user and his withheld community on the right. We study four user cases: two where LDA ranks better, as shown in Figure 6 (a) to (d), and two for the opposite, as shown in Figure 6 (e) to (h). All user data were anonymized for safeguarding user privacy. We can make three observations about the type of community, similarity between communities, and population of community.

First, consider users Doe#1 and Doe#2, for whom LDA ranks better than ARM. The topic distributions of their

Table 5: The community information for user Doe#1. The category of each community is defined on Orkut. Last community is the withheld community while the rest are joined communities.

Community #	Community Name	Category	Size
59641	"8085" microprocessor	Alumni/Schools	84
102299	windows vista	Computers/Internet	7041
18663	turbo C/C++ programming	Computers/Internet	198
104613	free books downloads	Computers/Internet	4190
95076	Novell	Computers/Internet	11
111431	Java reference	Computers/Internet	504
31820	ImageJ-Java image processing	Computers/Internet	1015
53474	web design	Computers/Internet	4598

Table 6: The community information for user Doe#2. Last community is the withheld community while the rest are joined communities.

Community #	Community Name	Category	Size
60640	Java certification	Computers/Internet	731
58344	professor Ayaz Isazadeh	Alumni/Schools	19
25422	persiancomputing	Computers/Internet	39
100953	Iranian J2EE developers	Computers/Internet	297
53474	web design	Computers/Internet	4598
27999	Yazd sampad	Schools/Education	17
43431	Tabriz university CS students	Alumni/Schools	13
80441	C#	Computers/Internet	2247
66948	Delphi	Computers/Internet	142

joined communities tend to be more *concentrated* (Figure 6 (a) and (c)), *i.e.*, they have low entropy. To uncover the semantics behind these topics, we summarize the users' community information, such as name, category and size, in Tables 5 and 6. We observe that these users are interested in communities related to Computer Technology. Even though the joined communities have different names, these users essentially have similar interests. On the contrary, consider users Doe#3 and Doe#4, for whom ARM ranks better. The topic distributions of their joined communities tend to be more *scattered* (Figure 6 (e) and (g)), *i.e.*, they have high entropy. We summarize these users' community information in Tables 7 and 8. Note that they have joined communities of different interests, such as Automotive, Culture, Food, Romance, and Education.

Second, when LDA ranks better, the withheld community and joined communities have a common set of highest-peak topics. For user Doe#1 (Figure 6 (a) and (b)), her withheld community 53474 overlaps with her joined communities at the peak topic 30. However, when ARM ranks better for a given user, there is no overlap at highest-peak topics between the withheld community and joined communities. Similarly, this is true for the topic distributions between a withheld community and its user. When LDA works better, user Doe#2 (Figure 6 (d)) has overlapped with his withheld community 66948 at peak topic 30; when ARM works better, user Doe#4 (Figure 6 (h)) has no overlap with his withheld community at peak topic 39.

Third, ARM ranks better for users whose communities are relatively large. For example, users Doe#3 and Doe#4 have relatively larger communities than users Doe#1 and Doe#2. This is perhaps because when a joined community is larger, it more frequently co-occurs with the withheld community. Hence, ARM can work better.

Overall, both LDA and ARM perform well for different types of users. LDA performs better for users who have joined relatively small communities of concentrated interests, and ARM is better for those who have joined relatively large communities of scattered interests.

Table 7: The community information for user Doe#3. Last community is the withheld community while the rest are joined communities.

Community #	Community Name	Category	Size
35049	Illegal street racin in KHI	Automotive	8526
95173	May 1985	Cultures/Community	381
3541	CCD-law college road	Food/Drink/Wine	650
14470	BMW enthusiasts	Automotive	32171
90730	17th May	Individuals	191
34939	I love to walk in the rain	Romance/Relationships	3895
116805	heights of laziness	Other	232
29789	Honda-owners and lovers	Automotive	4584

Table 8: The community information for user Doe#4. Last community is the withheld community while the rest are joined communities.

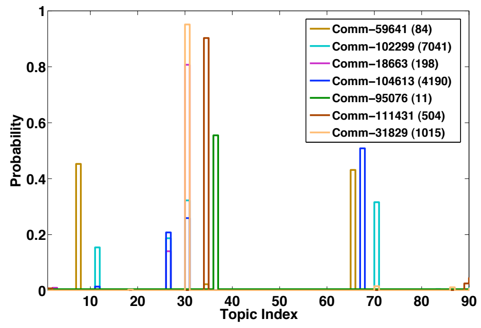
Community #	Community Name	Category	Size
50279	Shahrukh Khan fan club	Individuals	50857
44363	girl power	Religion/Beliefs	1467
109245	love never dies	Romance/Relationships	22600
111271	why friendz break our heart	Romance/Relationships	10301
38320	holy angels school	Alumni/Schools	95
15760	why life is so unpredictable	Other	3878
8886	T20 WC champs	Recreation/Sports	43662
77269	star-one fame serial-remix	Other	403
51302	left right left	Arts/Entertainment	13744
68215	life is too short to live	Other	8197

5.2 Scalability: Runtime Speedup in Distributed Environments

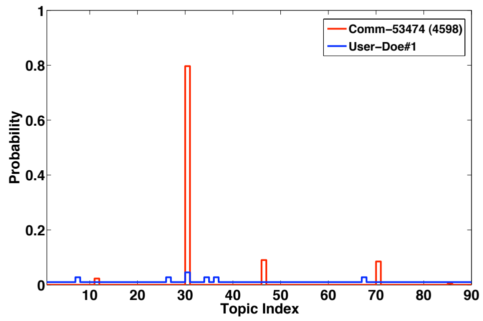
Having demonstrated LDA's promising performance for the community recommendation task, we then parallelize it to gain speedup. Our parallel LDA code was implemented in C++. To conduct our scalability experiments, we used the same Orkut data set as was used in Section 5.1. In analyzing the runtime speedup for parallel LDA, we trained LDA with 150 topics and 500 iterations. Our experiments were run on up to 32 machines at our distributed data centers. While not all machines are identical, each machine is configured with a CPU faster than 2GHz and memory larger than 4GB.

In Table 9 we report the speedup on the Orkut data set. We separate total running time into three parts: computation time, communication time and synchronization time. Communication time is incurred when message-passing takes place between machines. Synchronization time is incurred when the root machine waits for task completion on the slowest machine. Table 9 indicates that parallel LDA can achieve approximately linear speedup on up to 8 machines. After that, adding more machines yields diminishing returns. When we use 32 machines, the communication time takes up nearly half of the total running time. Hence, it is not worthwhile to add more machines after that.

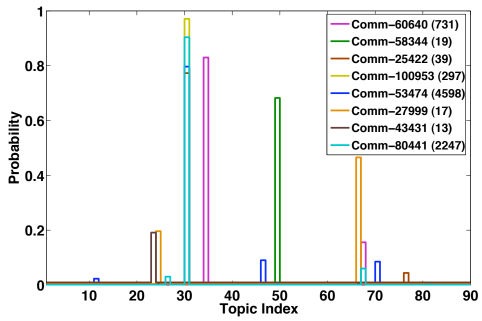
Figure 7 shows the speedup curves and overhead analysis. In the figure, we draw on the top the computation-only line (Comp), which approaches the linear speedup line. Computation speedup can become sublinear when adding machines beyond a threshold. Note that other jobs may be run simultaneously with ours on each machine, though we chose a data center with a light load. The result is expected due to Amdahl's law: the speedup of a parallel algorithms is limited by the time needed for the sequential fraction of the algorithm (*i.e.*, step 12 in Algorithm 1). When accounting for communication and synchronization overheads (the Comp + Comm line and the Comp + Comm + Sync line), the speedup deteriorates. Between the two overheads, the synchronization overhead has very little impact on the speedup compared to the communication overhead.



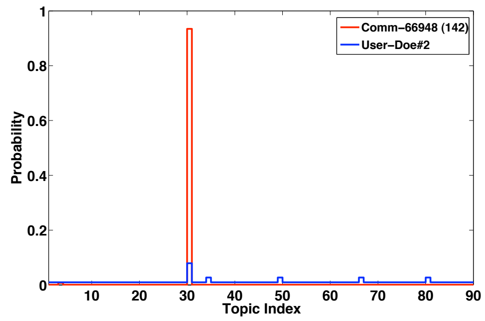
(a) Topic distributions of 7 joined communities for user Doe#1.



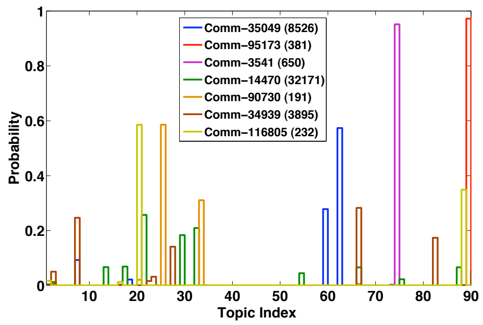
(b) Topic distributions for user Doe#1 and her withheld community 53474. The withheld community is ranked 1 by LDA and 438 by ARM.



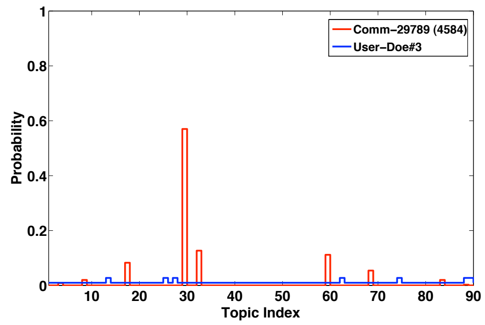
(c) Topic distributions of 8 joined communities for user Doe#2.



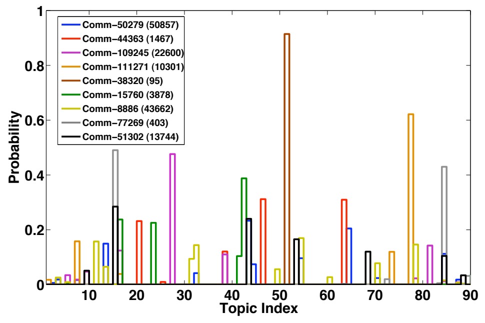
(d) Topic distributions for user Doe#2 and his withheld community 66948. The withheld community is ranked 11 by LDA and 449 by ARM.



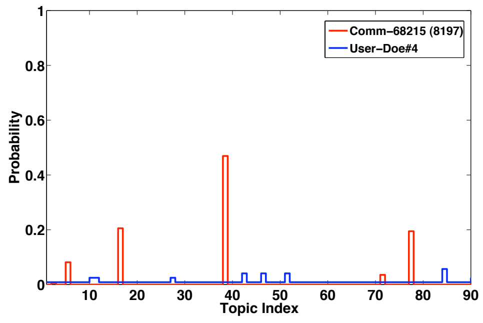
(e) Topic distributions of 7 joined communities for user Doe#3.



(f) Topic distributions for user Doe#3 and her withheld community 29789. The withheld community is ranked 8 by LDA and 1 by ARM.



(g) Topic distributions of 9 joined communities for user Doe#4.



(h) Topic distributions for user Doe#4 and his withheld community 68215. The withheld community is ranked 5 by LDA and 1 by ARM.

Figure 6: The topic distributions of joined communities (on the left) and withheld communities (on the right) for four users. We also graph the topic distribution for each user. The number in parentheses for each community is its size.

Table 9: Orkut data set. Runtime for LDA using different numbers of machines. 492,104 users, 118,002 communities, 150 topics, α=0.33, β=0.1.

Machines	Comp	Comm	Sync	Total	Speedup
1	28911s	0s	0s	28911s	1
2	14543s	417s	1s	14961s	1.93
4	7755s	686s	1s	8442s	3.42
8	4560s	949s	2s	5511s	5.25
16	2840s	1040s	1s	3881s	7.45
32	1553s	1158s	2s	2713s	10.66

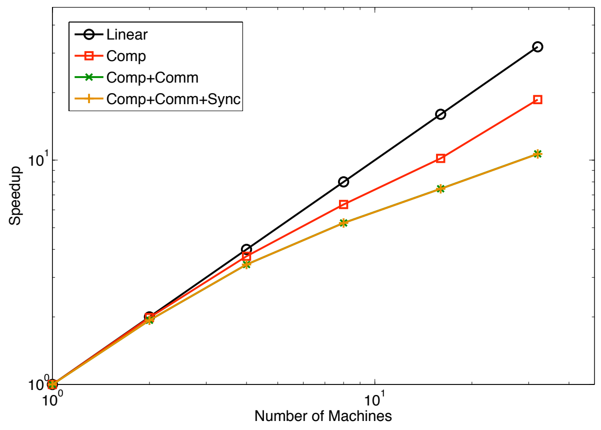


Figure 7: Speedup and overhead analysis.

6. CONCLUSIONS

In this paper, we have compared ARM and LDA for the community recommendation task, and evaluated their performances using the top-k recommendations metric. Our empirical comparisons using the top-k recommendations metric show a surprisingly intuitive finding: that LDA performs consistently better than ARM for the community recommendation task when recommending a list of 4 or more communities. However, for recommendation lists of up to 3 communities, ARM is still a bit better. We analyzed the latent information learned from LDA to illustrate why it is better at generating longer recommendation lists. To handle large-scale data sets efficiently, we parallelized LDA to take advantage of the distributed computing infrastructure of modern data centers [1]. Our scalability study on the same Orkut data set shows that our parallelization can reduce the training time from 8 hours to less than 46 minutes using up to 32 machines.

There are several directions for future research. First, our current user-community data is binary-valued to denote membership or non-membership (one or zero). We can consider extending our recommendation algorithms to handle integer-valued or real-valued relations. For example, to denote the strength of relationship between a user and a community, we can use either the number of posts from this user to that community's forum, or the number of visits by the user to the community's forum. Second, we can extend our ARM method to take multi-order rules into consideration rather than just first-order rules.

7. ACKNOWLEDGMENTS

The authors would like to thank Ellen Spertus for providing the Orkut data used in the experiments, and Feng Yan for his stimulating discussions and helpful comments.

8. REFERENCES

[1] Open source parallel lda. http://code.google.com/p/plda/.

[2] R. Agrawal, T. Imieliński, and A. Swami. Mining association rules between sets of items in large databases. In *Proc. of the 1993 ACM SIGMOD conference*, pages 207–216, 1993.

[3] R. Agrawal and R. Srikant. Fast algorithms for mining association rules in large databases. In *Proc. of the 20th VLDB conference*, pages 487–499, 1994.

[4] D. M. Blei and M. I. Jordan. Modeling annotated data. In *Proc. of the 26th ACM SIGIR conference*, pages 127–134, New York, NY, USA, 2003.

[5] D. M. Blei, A. Y. Ng, and M. I. Jordan. Latent Dirichlet allocation. *Journal of Machine Learning Research*, 3:993–1022, 2003.

[6] J. Dean and S. Ghemawat. Mapreduce: simplified data processing on large clusters. *Communications of the ACM*, 51(1):107–113, 2008.

[7] T. L. Griffiths and M. Steyvers. Finding scientific topics. *Proceedings of the National Academy Science*, 101 (suppl. 1):5228–5235, April 2004.

[8] W. Gropp, E. Lusk, and A. Skjellum. *Using MPI-2: Advanced Features of the Message-Passing Interface*. MIT Press, 1999.

[9] J. Han, J. Pei, Y. Yin, and R. Mao. Mining frequent patterns without candidate generation: A frequent-pattern tree approach. *Data Min. Knowl. Discov.*, 8(1):53–87, 2004.

[10] K. Järvelin and J. Kekäläinen. Cumulated gain-based evaluation of IR techniques. *ACM Transactions on Information Systems*, 20(4):422–446, 2002.

[11] Y. Koren. Factorization meets the neighborhood: a multifaceted collaborative filtering model. In *Proc. of the 14th ACM SIGKDD conference*, pages 426–434, 2008.

[12] H. Li, Y. Wang, D. Zhang, M. Zhang, and E. Y. Chang. Pfp: parallel fp-growth for query recommendation. In *Proc. of the 2008 ACM RecSys conference*, pages 107–114, 2008.

[13] A. McCallum, A. Corrada-Emmanuel, and X. Wang. The author-recipient-topic model for topic and role discovery in social networks: Experiments with enron and academic email. Technical report, Computer Science, University of Massachusetts Amherst, 2004.

[14] J. J. Sandvig, B. Mobasher, and R. Burke. Robustness of collaborative recommendation based on association rule mining. In *Proc. of the 2007 ACM Recommender Systems conference*, pages 105–112, New York, NY, USA, 2007. ACM.

[15] M.-L. Shyu, C. Haruechaiyasak, S.-C. Chen, and N. Zhao. Collaborative filtering by mining association rules from user access sequences. In *Proc. of the International workshop on Challenges in WIRI*, pages 128–135, 2005.

[16] M. Snir and S. Otto. *MPI-The Complete Reference: The MPI Core*. MIT Press, 1998.

[17] E. Spertus, M. Sahami, and O. Buyukkokten. Evaluating similarity measures: a large-scale study in the orkut social network. In *Proc. of the 11th ACM SIGKDD Conference*, pages 678–684, 2005.

[18] R. Srikant and R. Agrawal. Mining generalized association rules. *Future Gener. Comput. Syst.*, 13(2-3):161–180, 1997.

[19] M. Steyvers, P. Smyth, M. Rosen-Zvi, and T. Griffiths. Probabilistic author-topic models for information discovery. In *Proc. of the 10th ACM SIGKDD Conference*, pages 306–315, 2004.

[20] R. Thakur, R. Rabenseinfer, and W. Gropp. Optimization of collective communication operations in MPICH. *International Journal of High Performance Computing Applications*, 19(1):49–66, 2005.