Applied Artificial Intelligence, 22:331–351

ISSN: 0883-9514 print/1087-6545 online
DOI: 10.1080/08839510801972835

Taylor & Francis
Taylor & Francis Group

COLLABORATIVE WEB AGENT BASED ON FRIEND NETWORK

Byeong Man Kim[1], Qing Li[2], Adele E. Howe[3], and Yuanzhu Peter Chen[4]
[1]*Department of Computer Science, Kumoh National Institute of Technology, Gumi, Gyeongbuk, Korea*
[2]*School of Economic Information Engineering, Southwestern University of Finance and Economics, China & School of Informatics and Computing, Arizona State University, Tempe, Arizona, USA*
[3]*Department of Computer Science, Colorado State University, Fort Collins, Colorado, USA*
[4]*Department of Computer Science, Memorial University of Newfoundland, St. John's, Canada*

□ *Most existing web search tools do not utilize previous search experiences of others to improve performance for the current user. In this article, we present a collaborative web agent designed to enable across-user collaboration in web search and recommendation. We use our distributed collaborative filtering (CF) algorithm based on a P2P overlay network of autonomous agents. Experiments show that our proposed scheme is more scalable than traditional centralized CF filtering systems and alleviates the sparsity problem in distributed CF.*

INTRODUCTION

It has become increasingly difficult to find useful information on the web, due to its large size and unstructured nature. Many techniques have been proposed by researchers to help users to obtain needed information efficiently. Examples include portals, web search engines, and meta-searching methods. Portals, such as yahoo.com, sohu.com, sina.com and naver.com, provide communally useful information related to some topic (e.g., latest news and stock information). Search engines such as Google and AltaVista respond to specific, spontaneous information queries from users.

As the need for information increases, users are no longer satisfied with finding out information by prime search engines. They are less keen to input queries over and over and to view long lists of returned matchings.

This work was supported by the Korea Research Foundation Grant (KRF-2005-013-D00048).

Address correspondence to Byeong Man Kim, Kumoh National Institute of Technology, Department of Computer Science, 1 Yangho-dong, Gumi, Gyeongbuk, 730-701 Korea. E-mail: bmkim@kumoh.ac.kr

Sometimes, they want the system to automatically deliver their needed information periodically, such as daily news related to their preferences. Thus, personalized information delivery approaches are proposed to serve their needs using web agents and collaborative information systems. Web agents, residing in users' machines, help users search the web and perform personalized information filtering and management. For instance, Blue Squirrel's WebSeeker (Bluesquirrel) and Copernic (Copernic) connect to various search engines, monitor web pages for the changes, and schedule automatic searches. SurfAgent (Somlo and Howe 2003) is a personalized web search agent that recommends new documents to a user by generating queries from the user profile and submitting them to a search engine. Google Alerts (Googlealert) provides e-mail updates of the latest relevant information based on predefined preferences.

A major problem with most web agents is that they do not facilitate user collaboration, although it has the potential to improve information search quality and efficiency greatly. We can only receive information returned by our own specific query. In reality, people usually solicit recommendations from friends and colleagues to acquire knowledge.

Currently, a few studies have investigated the web collaborative agent, which combines the collaborative filtering and agent techniques to provide personalized information searching and recommendation. Balabanovic and Shoham (1997) propose "Fab," an adaptive web page recommendation service. This system takes advantage of the shared interests among users without losing the benefits of the representations provided by content analysis. Although it is an agent-based system, it needs a central repository to collect all the information, which greatly reduces the autonomy and flexibility of the agent system. Letizia (Liebermann, 1995) proactively follows links in a page being viewed, and chooses targets best matching a user profile learned by monitoring the user's behaviors. In Letizia, the agent presents its recommendations in a separate "channel surfing" window, which continuously displays recommendations related to a special topic. Such information channels are organized based on the opinions of a group of users who possess similar interests. In this case, users are encouraged to select their favorite channel to share the information with others.

In this article, we propose a web agent system, a distributed CF recommender based on peer-to-peer (P2P) networking. It enhances SurfAgent (Somlo and Howe 2003) by adding a collaborative recommendation function. It is able to provide collaborative recommendation and to search for long-term interests of users. SurfAgent learns from user profiles to construct more informative queries to meet the long-term interests of users. For instance, SurfAgent can help user Tom find his favorite music by converting a simple query of "pop music of Whitney Houston" to a more specific query of "Whitney Houston's songs on Billboard after 1990" by

analyzing Tom's profile. SurfAgent, however, does not make any serendipitous connections for Tom. If Tom has never heard songs of Mariah Carey, he will probably not indicate any interest in her songs in the query. To some audience, the styles of Whitney Houston and Mariah Carey may be quite similar. Therefore, there is a good chance for Tom to like the songs of Mariah Carey as well. Based on this fact, we extend SurfAgent by adding the collaborative recommendation function.

In our system, agents collaborate with each other by sharing their rating information on items with the user's friends, i.e., other users with similar preferences. The rating information, however, is not centralized on a certain server but distributed among agents' local computers. Each agent maintains its user's ratings and broadcasts them to the friends. That is, only the ratings of the user and friends are stored in an agent's local database. Based on this local information, the agent makes recommendations.

P2P systems are designed to be scalable in that, as more peers join the system, they improve their capabilities (Chawathe, Ratnasamy, and Breslau, 2003). We adopt a P2P approach to make our recommender system more scalable. Peers are agents and a Gnutella-like P2P protocol is used to discover friends.

The rest of the article, is organized as follows. The next section reviews related work on recommender systems and collaborative filtering. The following section introduces our system architecture and its typical operation. The section entitled "DISTRIBUTED CF" elaborates on the primary components of our system, i.e., a novel distributed CF approach. An experimental evaluation of our work is presented in the following section. In the section "DISCUSSION," we discuss the unique features of our framework in the context of similar works before we conclude the article.

RELATED WORK

Recently, it has been proven that AI-based techniques are useful in helping users to handle large amounts of information from the internet (Ferber 1999; Heflin 2001; Nick and Themis 2001). The ideas of personalized search engines, intelligent software agents, and recommender systems have been widely accepted among users who need advice during a search. In a relatively short time, a variety of recommender systems including agent systems have been developed (Resnick et al. 1994; Shardanand 1995; Wittenburg, Das, Hill, and Stead, 1995; Balabanovic and Shoham 1997; Breese, Heckerman, and Kardie 1998; Sarwar et al. 2001, 2002; Montaner, Lopez, De LaRosa, and Li 2003; Ali and Wijnand 2004; Han, Xie, and Shen 2004a; Hofmann 2004; Kim and Li 2004; Li, Kim, Guan, and Oh 2004; Pitsilis and Marshall 2005; Xue et al. 2005; Kim, Li, Park, and Kim 2006).

At the initial stage, a preliminary recommender system applies content-based filtering (CBF) to provide recommendation services. It returns appropriate results to users by matching user preferences against stored representations of searching information. The user preference can be provided either implicitly using a user profile or explicitly by the user. SurfAent (Somlo and Howe 2003), the basis of our work, is an example of these agent systems using CBF. Content-based filtering has been shown effective in locating textual items relevant to a topic. It does not, however, provide serendipitous recommendations because all the information is selected and recommended based on only contents.

Collaborative filtering (CF), developed by GroupLens (Resnick et al. 1994) and Ringo (Shardanand 1995) independently, debuted as a prevailing and efficient technique for recommendation systems. In CF, preferences of a given user are further employed to predict the interests of others. A target user is matched against the database to discover friends, which are essentially users with similar historical tastes. Items favored by friends are then recommended to the target user based on their similarities. Collaborative filtering-based techniques have been adopted rapidly both in academia and industry. Many CF-based systems have been proposed, such as MovieLens, a movie recommender (Breese et al. 1998), and group asynchronous browsing (GAB) (Montaner et al. 2003), a web page recommender using bookmarks. A growing number of entities, such as Amazon.com, CDNow.com, and Levis.com, include CF-based algorithms in their recommender systems.

Although CF can improve the quality of recommendations based on user ratings, it does not use any information that can be extracted from semantic contents. Thus, the quality of recommendation solely depends on the user similarities. Collaborative filtering also has the known problems of cold-start (Sarwar et al. 2001), sparsity (Han et al. 2004a; Hofmann 2004), and scalability (Shahabi, Banaei-Kashani, Chen, and McLeod 2001; Sarwar et al. 2002; Xue et al. 2005). Recently, some hybrid systems (Montaner et al. 2003; Li and Kim 2004; Kim et al. 2006) have been explored to cope with these shortcomings by combining CF with CBF. These proposals, however, are not suitable for the distributed environment as where SurfAgent is deployed. In particular, these CF computations are executed in one or several central servers that concentrate and analyze information from individual users for recommendations. Apparently, such an approach is not feasible in our settings because it is not practical for an agent in a web client to maintain the rating information of all users. Instead, our agents must make distributed recommendations using local machines rather than central servers for more autonomy and improved scalability.

To date, most of the research on CF algorithms focuses on how to make good recommendations efficiently, but the scalability problem has been

overlooked. Most existing approaches to scalable CF algorithms resemble P2P systems. P2P systems can be categorized into two groups. The distributed CF algorithms in the literature (Tveit 2001; Ratnasamy, Shenker, and Stoica 2002; Pitsilis and Marshall 2005) are based on either of those two paradigms. One model is exemplified by Gnutella (Chawathe et al. 2003). This type of P2P system has an unstructured overlay and uses limited message flooding to locate information. To find a target in the overlay, a peer sends a request to its neighbors. In turn, these neighbors forward the query to their neighbors until the request has traveled a certain range. Tveit (Ali and Wijnand 2004) takes this approach. The other type is based on distributed hash tables (DHTs) (Ratnasamy et al. 2002). In a DHT-based P2P system, each resource is associated with a key and each node stores a subset of resources and corresponding keys. A peer locates its intended resource by issuing a lookup request using a key. The node that stores the resource with the key returns its identity (e.g., the IP address). In these systems, peers are organized into a well-defined structure for efficient routing. PipeCF (Han, Xie, Yang, and Shen 2004b) takes this approach and is motivated by Plaxton et al. (Plaxton, Rajaraman, and Richa 1997). While DHT is efficient in a relatively stable P2P overlay, it is not as robust as an unstructured overlay when peers join and leave the system. Since our network of agents is fairly liberal for its members, we take the Gnutella-like approach.

SYSTEM ARCHITECTURE

In this article, we enhance SurfAgent (Somlo and Howe 2003) by adding a novel distributed CF module to provide serendipitous recommendations for multimedia data such as movies, pictures, music, etc. As illustrated in Figure 1, the system structure consists of two main parts: the web search (above the dotted line) and the collaborative recommender (below the dotted line). SurfAgent is essentially the top half. We add the collaborative recommendation functionality to complement SurfAgent. The new agent system is not only able to provide the information search service but also makes it possible to provide recommendation service for users, so that a user can benefit from previous search experiences of other users. A simple scenario of operation of our agent system is as follows:

i. Each agent maintains a profile for its own user representing search preferences.
ii. Once a user inputs a query, the agent refines the query based on the user profile. While the user is browsing the web pages retrieved by the system, implicit or explicit feedbacks are transferred by the monitor to update the user profile for further searches. Meanwhile, the rating information used for collaborative recommendation is also updated.

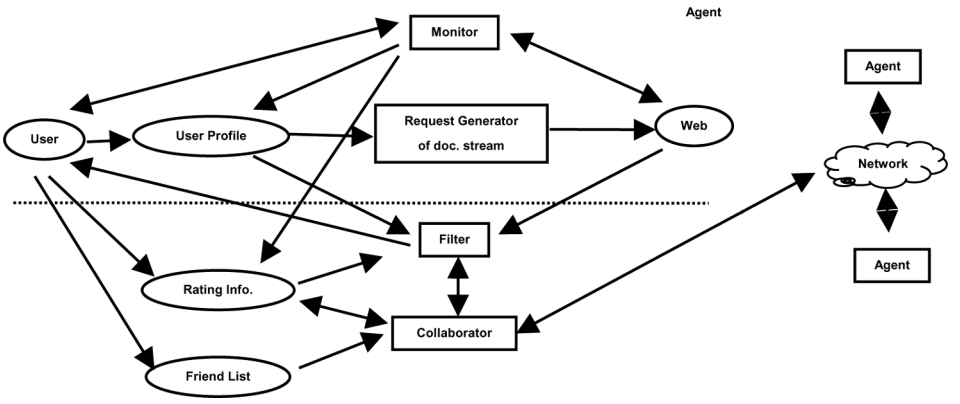

FIGURE 1 System architecture.

iii. At the same time, according to those continually updated ratings, the agent broadcasts the recommendation through its friend list. Then the friends will forward this recommendation to their friends and so forth. As a result, the recommendation is able to cover a large number of users.
iv. A user's agent also periodically receives recommendations from friends for new items such as URLs, audio, books, or video related to the preferences. In addition, the agent helps the user find out some useful information for long-term query services based on the user profile. (The long-term query service and query modification are described in Somlo and Howe (2003).)

DISTRIBUTED CF

In this section, we detail the technical aspects of our distributed CF recommendation system. Instead of monolithic repositories situated on central servers to provide recommendation services, we take a decentralized approach based on a P2P overlay network of autonomous agents.

In contrast to a centralized CF algorithm, where central servers collect and analyze the data from all users, a distributed CF algorithm assigns the task to individual agents and makes them find the nearest friends and receive recommendations from these friends. In our work, we propose a friend-link based routing algorithm to distribute prediction tasks to local agents featuring solutions to two well-known problems in the CF research area, i.e., scalability and sparsity.

Although it may not be difficult for a central server to handle hundreds or thousands of users, it will certainly be a performance bottleneck when the number of users in the system becomes larger. If each local client deals

only with its own recommendation, there is no need for a powerful center to process concentrated data, and the system will be scalable. This is verified by our experiment in that the proposed work provides a good scalability without bringing down the recommendation services in the next section.

Below we describe the four essential aspects of our recommender system:

(1) personal data creation and maintenance
(2) friend-link based routing
(3) local recommendation engine
(4) solution to some typical recommendation issues.

Personal Data Creation and Maintenance

In our system, each agent maintains a profile to record user preferences, collected implicitly via user visiting history or explicitly via user input. User preferences can be denoted by multiple preference vectors, each representing a favorite topic. For simplicity, in this article, we assume that user preferences are modeled by one preference vector. A subset of global ratings in the network is stored locally as well. That is, each agent not only keeps its own ratings on items but also those of friends on a friend list. The friend list contains IDs of friends and their rating information. The friend list is generated and updated as in subsequent subsections.

Keeping the database locally for each user has some advantages in a dynamic P2P environment.

i. Making off-line information available for real-time recommendation. Peers often join and leave the dynamic P2P environment so that the valuable user information may not be available all the time. With the replicas of their ratings, we can still use the information from off-line users.
ii. Reduction of network traffic at recommending time. In fact, the friend list functions as a shortcut to create additional links atop the P2P system overlay. These shortcuts are implemented as a separate performance enhancement layer over existing content location mechanisms, namely, flooding in Gnutella. The benefits are two-fold. First, shortcuts are modular in that they can work with any underlying content location scheme. Second, shortcuts only serve as performance-enhancement hints. If some friends on the list are not reachable, the local peer can locate new friends using the overlay. Therefore, having a shortcut layer created by the friend lists can reduce hop-by-hop delay in the overlay.

Friend-Link Based Routing Mechanism

Gnutella has a simple routing algorithm. A request is flooded to all peers in the system as shown in Figure 2(a). A peer stops rebroadcasting if the target is considered matched and returns a "found" message back to the broadcast originator. Otherwise, it decreases the TTL (time to live) by 1 and then passes on the request to its neighbors. If the TTL count reaches 0, the request is discarded.

The recommendation quality of our system undoubtedly depends on the discovery of friends. And the discovery process is illustrated in Figure 2(b). In order to find the friends for each peer at the beginning, the system initiates a flooding similar to Gnutella as follows. A request is broadcast to neighbors of the originating peer. The request contains the user preference vector of the originator. If the similarity of the preference vectors between the sender and receiver is greater than a threshold, the receiver sends its ID and ratings back to the originator. Otherwise, it forwards the request to its neighbors after decrementing TTL by 1. If the TTL reaches 0, the request is removed. When a large set of friends are returned to the originator, the returned peers are ranked based on the similarity. The top n peers are regarded as friends.

Once the friend list is constructed for each peer, a shortcut layer can be built based on the friendship between peers. We call it the "friend network." One peer can continually update his rating information and broadcast it to his friends. At the same time, each peer can receive the ratings from its friends and thus is able to update his database.

Things change; people change. As time goes by, a friend of a peer may no longer be similar. It is thus necessary to update each peer's friends list periodically. Instead of flooding the whole network as at the beginning, we trace the friend relationship to identify new friends. By requesting friends to recommend new friends, we can significantly reduce the friend

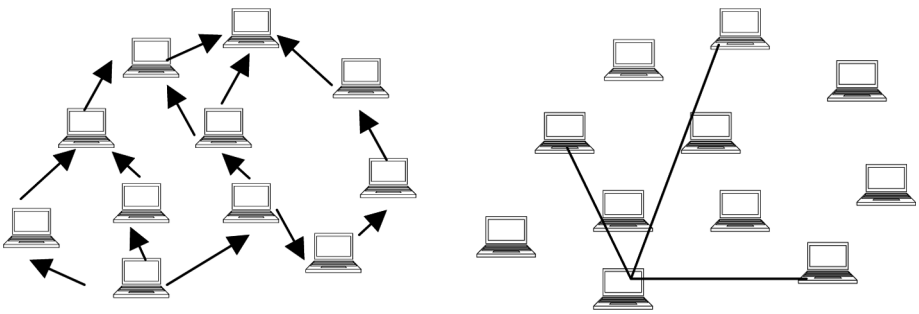

FIGURE 2 Routing.

searching range instead of flooding the network. However, if we send requests only to friends of the changing peer, we may possibly miss some potentially "friendly" nodes. To alleviate this situation, we introduce Friendship Depth To Live (FDTL), which is similar to TTL except that it operates on the friend network.

A request originated at a target (changing) peer, in this case, contains the rating information of the target peer for friendship calculation. Note that the preference vector is used for this purpose at the beginning because no rating information is available. If still no rating is given by the user at the time, the preference vector is included in the message instead of the rating information. Once a peer receives a request from a friend, it calculates the similarities between its friends and the target peer contained in the request. If this measure of a friend is greater than a threshold, the friend ID with its rating information is sent back to the target peer. In our system, we set the threshold to the lowest similarity on the current friend list. Thus, the likelihood that a peer suggests new friends to the target peer decreases as the system stabilizes. This leads to a decreased number of replies and consequently traffic in the system.

We call the peers suggested by a friend "potential friends." After the target peer has received replies, it selects the top n peers among its current friends and the potential friends, and then registers them as its new friends by updating its local database. This allows a peer to adapt to the dynamic changes and to gradually refine friend selection.

As an example, the following describes a simple scenario of our friend discovery and selection mechanism once each peer has created its own friends list. Assume that each peer keeps n friends in its friend list.

1. Agent A sends his A query containing its ratings on items to its friends on the friends list. The query is tagged with a FDTL. In Figure 3, we can cover all peers if the FDTL of peer A is set 3.

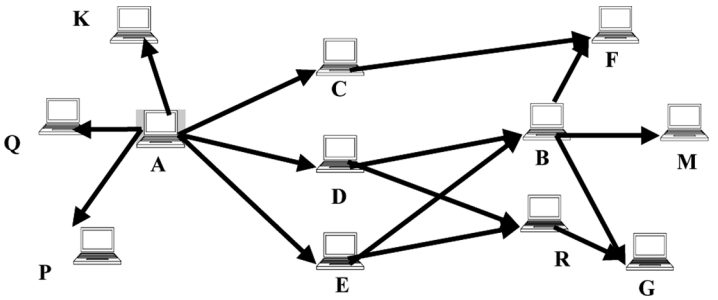

FIGURE 3 Friend network.

2. Once a friend, say D, receives A's query, D will recommend potential friends of A by comparing A's ratings with those of D's friends. Then D will send back the IDs and ratings of the potential friends whose similarities are greater than the threshold.
3. Afterwards, agent D will continue to forward A's query to its friends, B and R. In turn, peer B sends A's query to its friends M and G.
4. Once agent A receives all the feedbacks from his query, it will update its friends list based on the new information.

In order to calculate the similarity between two users, we use two measures. One is the cosine similarity between preference vectors, which is commonly used in information retrieval and text-mining (Sarwar et al. 2001). The similarity between two users k and l in this case is defined as

$$sim(k, l) = sim(D_1, D_2) = \frac{\sum_i x_{1i} x_{2i}}{\sqrt{\sum_j x_{1j}^2}\sqrt{\sum_j x_{2j}^2}} \quad (1)$$

where D_1 and D_2 are the preference vectors of k and l, and x_{1i} and x_{2i} denote the ith component of D_1 and D_2, respectively.

The other measure is the Pearson correlation (Resnick et al. 1994), which represents the degree to which a linear relationship exists between two vectors. Here, to give credit to the friends who share more common ratings with the local peer, we adopt a variant of the Pearson correlation

$$sim(k, l) = \frac{\max(|V_k \cap V_l|, \beta)}{\beta} \times \frac{\sum_{i=1}^{m} (R_{k,i} - \overline{R}_k)(R_{l,i} - \overline{R}_l)}{\sqrt{\sum_{i=1}^{m} (R_{k,i} - \overline{R}_k)^2}\sqrt{\sum_{i=1}^{m} (R_{l,i} - \overline{R}_l)^2}} \quad (2)$$

where $sim(k, l)$ is the similarity between users k and l, β is the threshold, and $|V_k \cap V_l|$ denotes the number of common ratings on the same items made by users k and l. Further, m is the total number of items that both user k and l rate on, $\overline{R}_k$ and $\overline{R}_l$ are the average ratings of users k and l, respectively, and $R_{k,i}$ and $R_{l,i}$ denote the ratings of users k and l on item i, respectively.

Local Recommendation Engine

To provide recommendation services to a user, an agent must predict the degree to which the user likes an item using the rating information in the local database. We consider two types of prediction here. One is to use the similarity between users, and the other is to use the similarity between items. The former is adopted in user-based CF algorithms and

the latter in item-based CF algorithms. Sarwar et al. (2001) show that item-based CF algorithms provide better quality of recommendation using complete information. However, our local database maintains only the rating of a user and its friends, and so only partial rating information is available. This can cause error in similarity calculation. Thus, we adopt the user-based CF in our approach.

Prediction for an item is calculated by performing a weighted average of deviations from the friend's mean. Here we use the top n rules to select the nearest N friends. The generic formula (Sarwar et al. 2001; Kim et al. 2006) for prediction on item i by user k is

$$P_{k,i} = \overline{R_k} + \frac{\Sigma_{l=1}^{n}(R_{l,i} - \overline{R}_l) \times sim(k, l)}{\Sigma_{l=1}^{n}|sim(k, l)|} \tag{3}$$

where *sim(k, l)* denotes the Pearson correlation similarity between users k and l, $P_{k,i}$ denotes the predication for the user k on item i, n is the number of friends of a user, $R_{l,i}$ denotes the rating of user l on item i, $\overline{R}_k$ denotes the average rating of user k on items, and $\overline{R}_l$ denotes the average rating of user l on items.

Issues with Recommendation Systems

The proposed distributed CF framework not only provides an answer to the scalability problem but also builds a bridge to the solutions of several other noticeable problems including "cold start," "sparsity," and "local optimal" problems.

The "cold start problem" (Sarwar et al. 2001; Kim et al. 2006; Li et al., 2007) is the case where the system cannot provide recommendations to new users with no history, i.e., no user ratings on items. Without user history, a CF system cannot find friends for a new user and, consequently, no recommendation can be generated. This is not a problem in our system. Since each user has his or her own profile in the system, we can find the friends of new users based on the content matching. With these friends, we can obtain predictions for the new user using Equation (3) with the following modification. In this equation, $\overline{R}_k$ is the average rating of user k on items. For a new user without any ratings on items, $\overline{R}_k$ should be the zero. Since $\overline{R}_k$ is the standard baseline of the user ratings and it is zero for a new user, it is not appropriate for us to apply this equation to a new user. Therefore, for a new user, we use $\overline{R}_{friends}$, the average of all ratings of the new user's friends, instead of $\overline{R}_k$.

The sparsity problem occurs when available data are insufficient for identifying similar users (friends). It is a major issue that limits the quality

	I1	I2	I3	I4	I5	I6	I7	I8
A		3		5			4	
B	1		3			5		
C			4					5
D	2				3			

FIGURE 4 Rating information of peer A.

of recommendation and the applicability of CF in general. To date, several approaches (Sarwar et al. 2001; Kim et al. 2006) have been proposed to alleviate this problem. However, it is inadequate to apply such an approach directly to our system because the rating information is not kept in central hosts. Here, we use a novel approach that borrows other agents' predictions instead of real ratings, called "pseudo ratings." For example, consider the local database of user A in Figure 4 and assume that the agent wants to predict the rating of user A on item I3. Since none of A's friends have any common item with A, the classic CF cannot do anything. Our approach overcomes this by requesting A's friends (B, C, and D) to make predictions

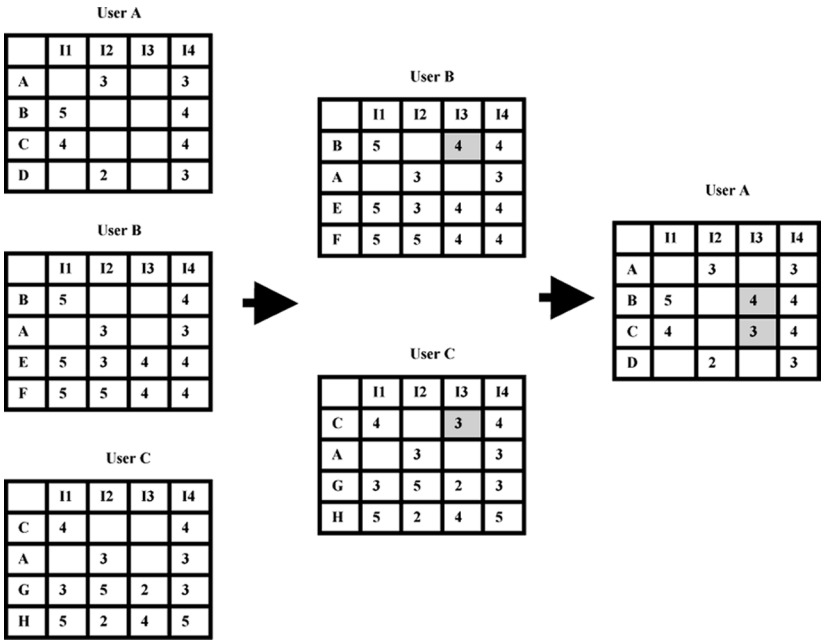

FIGURE 5 Recommendation from friends.

on item I2, I4, and I7. On a prediction request, a friend agent calculates the prediction value using Equation (3) and returns it to the requestor.

Although the "local predication" problem is unique in the P2P CF framework, the pseudo ratings provide a good approximation to it. For instance, as shown in Figure 5, user *A* does not have a friend with a rating on item *I3*. Item *I3* is treated like a new item though it is actually not and, thus, agent *A* cannot make any prediction on it. However, maintaining a friends list enables an agent to ask a friend's opinion when rating information in the local database is not sufficient for prediction. Some friends can deduce their opinions (pseudo rating) on an item based on their friends' ratings. In Figure 5, users *B* and *C* make predications on item *I3* based on their friends' rating (the prediction value are shaded), respectively. Therefore, user A is able to obtain an approximate prediction based on user *B*'s and user *C*'s opinions on item *I3*.

PERFORMANCE EVALUATION

Our proposed distributed CF recommender system decentralizes the recommendation services to peers in the network to battle the scalability problem. And this is attained without noticeable degradation of the prediction quality provided, verified by our experiments.

We carried out experiments using the Each-Movie data, collected by the Digital Equipment Research Center. There are 1623 movies and 61,265 users with a total of over 2.8 million ratings in the data set. The ratings were explicitly entered by users and are integers between 0 and 5. This data set is publicly available for collaborative filtering testing. Before presenting our experimental results, we first define the prediction accuracy to be used as a metric of the recommendation quality.

Prediction Accuracy

In order to evaluate the recommendation performance, we use the mean absolute error (MAE) as the accuracy metric. Mean absolute error (Breese et al. 1998) has been used widely in evaluating the accuracy of a recommender system. It is calculated by averaging the absolute errors in rating-prediction pairs as follows:

$$MAE = \frac{\Sigma_{j=1}^{N} |P_{i,j} - R_{i,j}|}{N} \tag{4}$$

where $P_{i,j}$ is system's rating (prediction) of item j for user i, and $R_{i,j}$ is the rating of user i for item j in the test data. Further, N is the number of rating-prediction pairs. A lower MAE denotes a greater accuracy.

Simulation

In order to simulate the initial environment, we select 1000 users, each with at least 20 ratings on movies in the data set. We calculate the similarity among them and maintain the 30 nearest friends for everyone. Then we randomly select 0–10 friends for each user. It simulates the beginning stage of the system. At this point, some users would like to select his friends from the database or the system selects someone as his friends based on the content analysis of the preference in user profiles. As reported by Breese et al. (1998), the recommendation performance depends on the number of friends. Fewer friends means less information available for recommendation. On the other hand, if the number of friends is over the optimum, it will add noise to affect the performance negatively. Based on our previous work (Li and Kim 2004; Kim et al. 2006), we set the friend number to 30 in our testing system.

We first randomly select 100 users and let them send out requests to their friends if any. These nodes are treated as seeds that spread the activity over the 1000 users and stimulate them to find their top 30 friends. After the first stage, the 100 seeds and their friends receive information to update their friends lists. However, the rest keep quiet and do not participate in the first iteration. In order to get these silent users involved, we make half of the remaining silent users send routing requests in the next iteration and so on. If the number of the silent users is less than 20, all of them are activated. As shown in Figure 6, the recommendation quality increases quickly during the earlier iterations, and stabilizes after about 25 iterations. If most users in a friends network cannot update any friends, we call this state "saturated," which is the optimal state of the network. As in

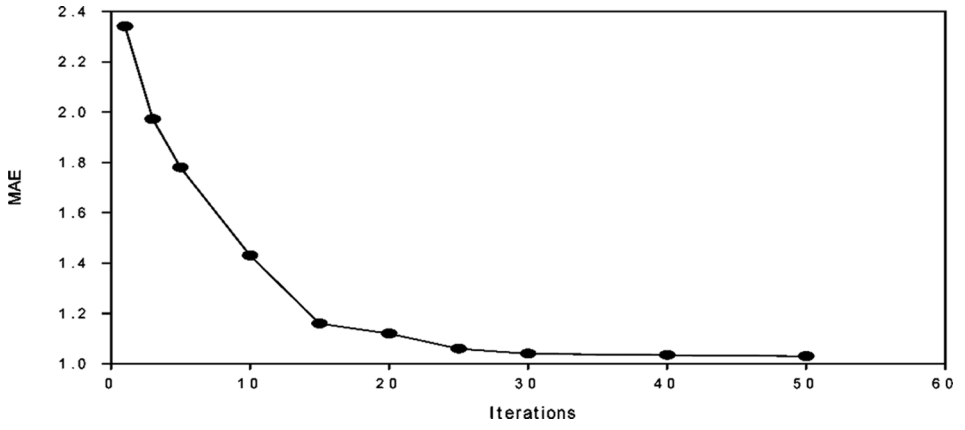

FIGURE 6 Recommendation quality vs. iterations.

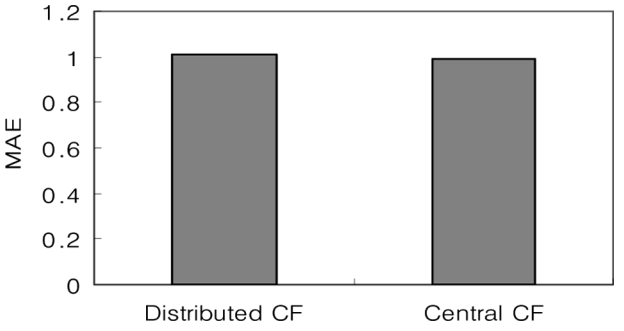

FIGURE 7 Accuracy: Distributed CF vs. centralized CF.

Figure 6, after this point, the users can no longer update their friends list based on the current situation.

We conduct an experiment to compare the prediction accuracies of the centralized and distributed CF algorithms. As shown in Figure 7, the MAE of distributed CF at the saturated point is quite close to the centralized CF. Remember that the distributed CF algorithm only uses the top 30 friends out of 1000 users. We use this information for finding the closest of an active user in the centralized CF algorithm.

The tremendous growth in the amount of available information and the number of visitors to websites in recent years poses some key challenges for recommender systems. One of them is achieving high coverage in the face of data sparsity without loss of accuracy. Here, the coverage is the ratio of the number of items predicated in the test data to the number of items containing ratings in the test data. To show how our distributed CF algorithm responds to this challenge, we compare its coverage with the centralized CF algorithm. As shown in Figure 8, the coverage of our algorithm is greater than the centralized one. Such an improvement is attributed to

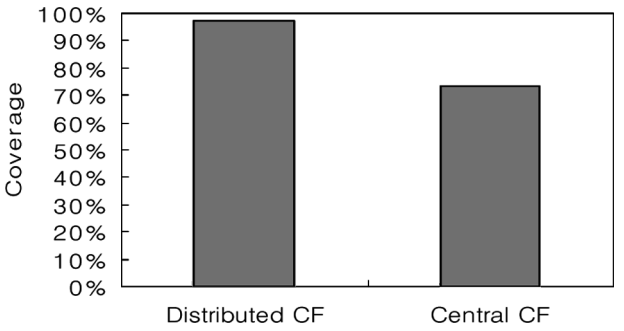

FIGURE 8 Coverage: Distributed CF vs. centralized CF.

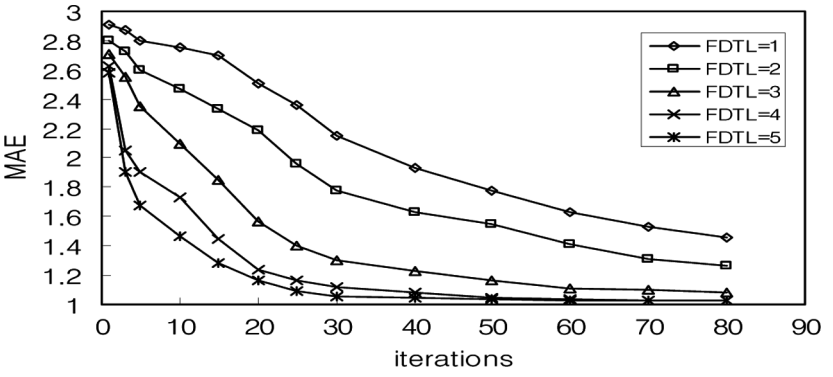

FIGURE 9 Recommendation performance over FDTL values.

the ability of the distributed CF algorithm in addressing the sparsity problem.

The value of FDTL has a direct impact on the recommendation quality. Apparently, the upper limit of FDTL affects the system performance as well. Figure 9 presents the relationship between the FDTL cap and the recommendation accuracy. As shown in Figure 9, when FDTL is set to 4, the system converges the fastest.

To simulate the dynamic environment, we build the optimal FDTL environment based on 80% of the ratings of the 1000 users, and then we add the remainder of the ratings of users to simulate the dynamic preference changes of users as time goes by. As shown in Figure 10, with the FDTL mechanism, the whole system can quickly propagate the new information to the target users and thus a quick response to the dynamic change.

To show the usefulness of the target-oriented path selection in P2P environments, two cases are investigated in our experiments, i.e., when the friend updating procedure is based on a friend network using FDTL

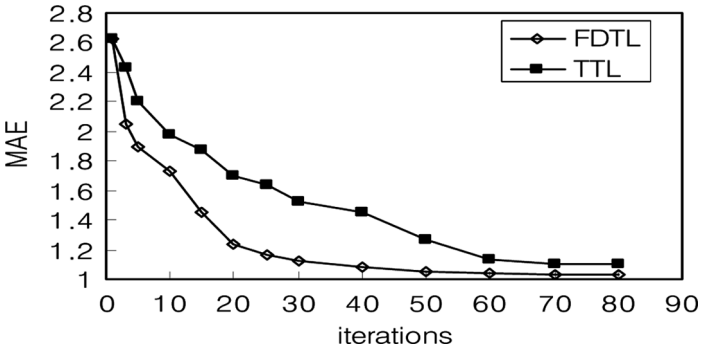

FIGURE 10 FDTL vs. TTL.

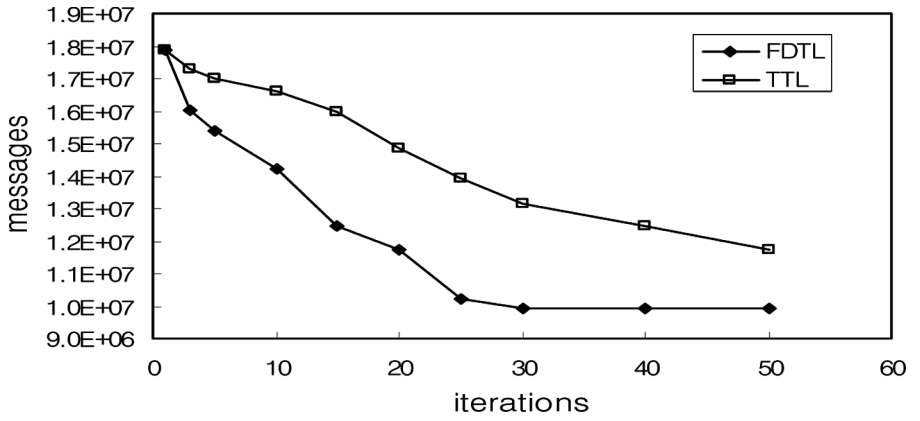

FIGURE 11 Updating overload.

and when based on an overlay network using TTL. In the latter case, 1–50 neighbors are randomly selected for each peer to construct a P2P network. We set TTL and FDTL to the same value of 4. The results are depicted in Figure 10. For more extensive simulations, we need a more complicated method to construct the P2P overlay. This is currently out of scope of this article, but a simple one as above is enough to show the usefulness of FDTL.

Since we keep the friends' ratings locally, our approach does not generate network traffic at recommendation time. However, our proposal generates network traffic to update friends' rating. To study the overhead of updating information, we record the number of "request" and "reply" messages used. As shown in Figure 11, the network traffic tends to decrease when the friend network stabilizes. This is due to the fact that it is rare to find any node that is greater than the threshold and should be sent back as "reply" after the system stabilizes. We also designed another experiment to study the communication overhead when TTL is used instead of FDTL under the same conditions as Figure 10. In fact, the method using TTL in Figure 10 and Figure 11 represents a typical Gnutella-like scheme. We observe that our FDTL-based approach has a smaller communication overhead.

For the cold start problem, since new users do not have any rating, traditional CF cannot find friends a new user. In our approach, we keep a preference vector for each user profile. Therefore, it can find friends for new users based on the similarity between the preference vectors. In our simulation, we use our previous method (Li and Kim 2004) to create preference vectors of users. As shown in Figure 12, which was started with the help of preference vectors, new users are able to continue updating their friends list and thus improve the predication on items. In Figure 12, we randomly

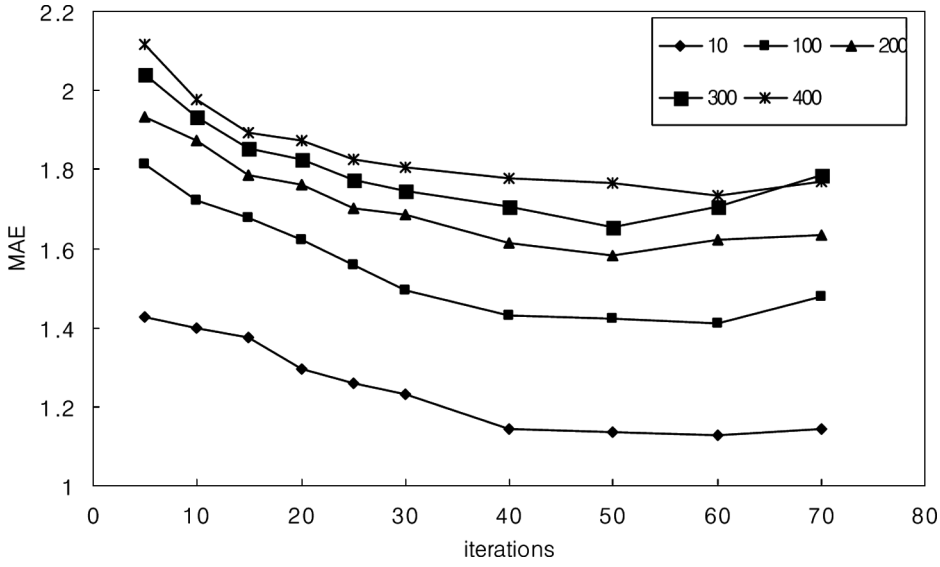

FIGURE 12 Cold start problem.

pick 10, 100, 200, 300, 400 users and delete their ratings and treat them as new users. We repeat the procedure 10 times. The reported numbers are the performance averaged over the 10 repetitions. It can be observed that the recommendation accuracy of new users is less than the average of the old users. This is because there is less information provided by new users than the old ones. However, our approach does provide a way to reflect the preferences of these new users.

DISCUSSION

Han et al. (2004a; 2004b) provided a distributed CF algorithm, PipeCF. Different from ours, they implemented the PipeCF on a P2P overlay using DHT. However, the performance of DHTs under the dynamic conditions common for P2P systems is unknown.

Tveit (2001) suggested a P2P CF system for mobile commerce based on the Gnutella schema. Peers are represented as software assistant agents interfacing with a mobile customer. When a peer receives a query rating vector, it calculates the similarity to the rating vector in the message and caches the messages at the node. If the similarity is greater than a threshold, the cached voting vector is sent back to the peer that sent the query rating vector. Otherwise, the query rating vector is broadcast further than its neighbors. It is noticed that Tveit's system has the communication overhead as any P2P network with an unstructured overlay.

TiVo (Ali and Wijnand 2004) television's CF system has been used by over 1 million clients for 4 years. TiVo applies an item-item form of CF which obviates the need for keeping any persistent memory of each user's viewing preferences at the server. This system adopted the client-server architecture. Such a design typically attempts to move the workload from the server to the clients to avoid a bottleneck. Although there were some strategies in the system to calculate the show-show correlation, the server had to do all those correlation estimations and to send them to the destination clients, which still leaves the scalability problem unattended.

All of those aforementioned systems do not have a special mechanism for the cold start problem, which is a critical problem for a practical system. However, our suggested approach provides a bypass of the cold start problem via user profiles. Although it is not capable proving the most accurate recommendations to users at the beginning, it does bootstrap the system quite well for newcomers.

Our proposed distributed CF algorithm is based on the Gnutella schema. The Gnutella schema typically provides better performance in a dynamic condition than DHT. This distinguishes our approach from PipeCF. The Gnutella schema may suffer from the scalability problem due to its flooding routing mechanism. We provide a self-organizing protocol by using friend lists to create shortcuts among peers, which provides a logical network on top of Gnutella's unstructured overlay. This feature makes our approach over any straightforward use of the Gnutella schema such as Tveit.

The recommendation routing in this work may seem similar to the query routing for unstructured P2P network. There is, however, a significant difference in that user nodes are able to be self-organized based on their degree of interests. This can greatly speed up calculation and improve the recommendation performance.

In our P2P recommendation structure, the friend list of an agent may not change rapidly as the system stabilizes. Thus, the collected friend information is useful to different recommendation requests. For example, if node *A* originates two different recommendation requests for item 1 and item 2, respectively, the stored friend information can be used for both requests. However, the generic P2P structure for query routing does not because most of query routing proposals retain the "blind" nature of query forwarding. In other words, the forwarding of queries is independent of the characteristics of the query and does not exploit the information contained in the query itself. Such proposals include blind flooding with a random-walk (Ratnasamy et al. 2002; Gkantsidis, Mihail, and Saberi 2004), expanding ring search reflecting the capacities of heterogeneous nodes in topology-construction (Chawathe et al. 2003), and caching pointers to content located one hop away (Chawathe et al. 2003). However, our routing

algorithm is guided by the content of user ratings instead of a "blind" way and leads to an effective routing in the P2P environment.

CONCLUSIONS

We proposed a distributed approach to enhancing a personal agent with a collaborative recommendation function, where an agent sends the user's rating information to its friends according to a self-organizing protocol. Our approach is more scalable than a centralized approach because an agent only keeps the rating information of itself and of its friends. From our experiments, we find that the accuracy of our approach is almost the same as a centralized CF approach using a subset of rating information. We also find that the coverage of our approach is better than a centralized one. This is due to the use of friends' opinions when rating information is not sufficient.

Distributing the work into a large number of clients addresses the scalability problem for large-scale applications. However, it may disclose sensitive private information to a wide scope. The protection of privacy in this regard will be part of our future work.

REFERENCES

Ali, K. and Wijnand, V.-S. 2004. TiVo: Making show recommendations using a distributed collaborative filtering architecture. *Proc. of the International Conference on Knowledge Discovery and Data Mining (KDD)*, Seattle, WA.

Balabanovic, M. and Y. Shoham. 1997. Fab: Content-based, collaborative recommendation. *Communications of the ACM* 40(3):66–72.

Bluesquirrel. www.bluesquirrel.com. accessed September 21, 2007.

Breese, J. S., D. Heckerman, and C. Kardie. 1998. Empirical analysis of predictive algorithms for collaborative filtering. *Proc. of UAI 98*, Madison, WI.

Chawathe, Y., S. Ratnasamy, and L. Breslau. 2003. Making Gnutella-like P2P systems Scalable. *Proc. of ACM Special Interest Group on Data Communication (SIGCOMM)*, New York, NY.

Copernic. www.copernic. com.

Ferber, J. 1999. *Multi-Agent Systems: An Introduction to Distributed Artificial Intelligence*. Boston, MA: Addison-Wesley Longman Publishing.

Gkantsidis, C., M. Mihail, and A. Saberi. 2004. Random walks in peer-to-peer networks. *Proceedings of IEEE Infocom*, Hong Kong, PR China.

Googlealert. www.google.com/alerts.

Han, P., P. F. Y. Xie, and R. Shen. 2004a. A novel distributed collaborative filtering algorithm and its implementation on P2P overlay network. *Proc. of The Pacific-Asia Conference on Knowledge Discovery and Data mining (PAKDD)*, Sydney, Australia.

Han, P., B. Xie, F. Yang, and R. Shen. 2004b. A scalable P2P recommender system based on distributed collaborative filtering. *Expert Systems With Applications* 27(2):203–210.

Heflin, J. 2001. Towards the semantic web: Knowledge representation in a dynamic, distributed environment. PhD thesis, University of Maryland, College Park, MD.

Hofmann, H. 2004. Latent semantic models for collaborative filtering. *ACM Transactions on Information Systems* 22(1):89–115.

Kim, B. M. and Q. Li. 2004. Probabilistic model estimation for collaborative filtering based on items attributes. *Proc. of IEEE\ACM Web Intelligence*, Washington, DC, USA.

Kim, B. M., Q. Li, C. S. Park, and S. Kim. 2006. A new approach for combining content-based and collaborative filters. *Journal of Intelligent Information Systems* 21(1):79–91.

Li, Q. and B. M. Kim. 2004. Constructing user profiles for collaborative recommender system. *Lecture Notes in Computer Science (LNCS)*. 3007:100–110.

Li, Q., B. M. Kim, D. H. Guan, and D. W. Oh. 2004. A music recommender based on audio features. *Proc. of ACM SIGIR*, Sheffield, UK.

Li, Q., S. H. Myaeng, and B. M. Kim. 2007. A probabilistic music recommender considering user opinions and audio features. *Information Processing & Management Journal (IP&M)* 43(2):473–487.

Liebermann, H. 1995. Letizia: An agent that assists web browsing. *Proc. of Intl. Conf. on AI.*

Montaner, M., B. Lopez, and J. L. De La Rosa. 2003. A taxonomy of recommender agents on the Internet. *Artificial Intelligence Review* 19, Montreal, Canada.

Nick, Z. Z. and P. Themis. 2001. Web search using a genetic algorithm. *Internet Computing, IEEE* 5(2): 18–26.

Pitsilis, C. and L. Marshall. 2005. A proposal for trust-enabled P2P recommendation systems. *Technical Report Series CS-TR-910*, University of Newcastle upon Tyne.

Plaxton, C. G., R. Rajaraman, and A. W. Richa. 1997. Accessing nearby copies of replicated objects in a distributed environment. *Proc. of Ninth Annual ACM Symposium on Parallel Algorithms and Architectures*, Newport, RI.

Ratnasamy, S., S. Shenker, and I. Stoica. 2002. Routing algorithms for DHTs: Some open questions. *Proc. IPTPS*, Cambridge, MA.

Resnick, P., N. Iacovou, M. Suchak, P. Bergstrom, and J. Riedl. 1994. An open architecture for collaborative filtering of netnews. *ACM Conference on Computer Supported Cooperative Work*, Chapel Hill, NC.

Sarwar, B., G. Karypis, J. Konstan, and J. Reidl. 2001. Item-based collaborative filtering recommendation algorithms. *Proc. of WWW Conference*, pp. 285–295.

Sarwar, B., G. Karypis, J. Konstan, and J. Riedl. 2002. Incremental singular value decomposition algorithms for highly scalable recommender systems. *Proc. of the 5th International Conference in Computers and Information Technology*, Dhaka, Bangladesh.

Shahabi, C., F. Banaei-Kashani, Y. S. Chen, and D. McLeod. 2001. Yoda: An accurate and scalable web-based recommendation system. *Proc. of the Sixth International Conference on Cooperative Information Systems (CoopIS 2001)*, Trento, Italy.

Shardanand, U. 1995. Social information filtering: Algorithms for automating "word of mouth." *Proc. of Computer-Human Interaction Conference (CHI)*, Denver, Co.

Somlo, G. L. and A. E. Howe. 2003. Using web helper agent profiles in query generation. *Proc. of the Second International Joint Conference on Autonomous Agents and Multiagent Systems*, pp. 812–818.

Tveit, A. 2001. Peer-to-peer based recommendations for mobile commerce. *Proc. of the 1st International Workshop on Mobile Commerce*, pp. 26–29.

Wittenburg, K., D. Das, W. Hill, and L. Stead. 1995. Group asynchronous browsing on the world wide web. *Proc. of the 4th WWW*, Boston, MA.

Xue, G. R., C. Lin, Q. Yang, W. S. Xi, H. J. Zeng, Y. Yu, and Z. Chen. 2005. Scalable collaborative filtering using cluster-based smoothing. *Proceedings of the 28th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, pp. 114–121.