A Decentralized CF Approach Based on Cooperative Agents

Byeong Man Kim
Kumoh National Institute of Technology
Gumi, Gyeongbuk, Korea
bmkim@kumoh.ac.kr

Qing Li
Information & Communications University
Daejeon, Korea
liqing@icu.ac.kr

Adele E. Howe
Colorado State University
Fort Collins, Colorado, USA
howe@cs.colostate.edu

ABSTRACT
In this paper, we propose a decentralized collaborative filtering (CF) approach based on P2P overlay network for the autonomous agents' environment. Experiments show that our approach is more scalable than traditional centralized CF filtering systems and alleviates the sparsity problem in distributed CF.

Categories and Subject Descriptors
H.3.3 [**Information Storage and Retrieval**]: Information Search and Retrieval –*information filtering.*

General Terms
Algorithms, Experimentation, Performance.

Keywords
Distributed collaborative filtering, P2P system, Friend network.

1. INTRODUCTION
To date, most of the research for CF algorithms was focused on how to efficiently make good recommendations. Even though touched on by some papers [1, 3, 6], the issue of solving the scalability problem by applying a distributed CF algorithm is not well studied.

We propose a novel distributed approach in which agents collaborate by sharing their rating information on items with their friends. Each agent saves its user's ratings and broadcasts them to the user's friends so that only friends' ratings and its own ratings are kept in the local database. Based on this local information, an agent makes recommendations.

We adopt a P2P approach [2] to make our system scalable. Peers are represented as the agents and a Gnutella-like P2P protocol is used to find friends at the beginning. After construction of the friend list, an agent operates by our protocol, which is almost the same as the Gnutella protocol [2], except that it is defined on a friend network instead of a P2P network.

2. OUR DISTRIBUTED CF APPROACH
Instead of monolithic repositories situated on central servers to provide the recommendation service, we use a decentralized collaborative filtering algorithm based on a P2P overlay network for the autonomous agents' environment. Each peer keeps his own user profile for each category of items (e.g., movies, music, Web pages, etc.). A user profile has user preferences which are collected implicitly via user visiting history or explicitly declared by the user. User preferences can be denoted by multiple preference vectors. For simplicity, we assume that user preferences are modeled by one preference vector. A subset of global ratings is also saved in user profiles. Each peer does not only keep its own ratings on items but also keeps a friend list which records the peers who share similar interests on items with the local peer. The friend list contains friend IDs and their rating information.

WWW 2006, May 23–26, 2006, Edinburgh, Scotland.
ACM 1-59593-323-9/06/0005.

2.1 Maintenance of Friend List
In order to find the friends for each peer in the network at the beginning, the system initiates a flooding mechanism similar to Gnutella as follows. A request is broadcast to neighbors of the sending peer. The request contains the user preference vector of the sender on items. TTL (time to live) is set in the original request package to limit the nodes to pass on. If the Cosine similarity of the preference vectors between neighbors and sender is larger than a threshold, it sends the neighbor ID and ratings back. Otherwise, it continues to forward the request to the neighbors, and TTL is decreased by 1. If the TTL count reaches 0, the request is not sent further. When a large set of friends is returned to the sender, all the returned peers are ranked based on the similarity of friendship. Top *n* peers are kept as friends; the rest are discarded.

Once a 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 continuously update his rating information and broadcast to his closest friends instead of his neighbors. At the same time, each peer can receive the ratings from his friends and thus is able to update his predictions.

Things change; people change. As time passes by, a friend of one peer may no longer be similar. It is necessary to update each peer's friend list periodically to get the up-to-date recommendations. 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 shorten the friend searching range instead of flooding the request in the Gnutella network. However if we send request messages only to friends of the target peer, we might miss some useful friends. To alleviate this situation, we introduce FDTL (Friendship Depth To Live) which is the same as TTL except that it operates on the friend network.

A request message contains rating information of the sender for calculation of friendship. 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 that time, the preference vector is included into the message instead of rating information. Once a peer receives a request message from its friend, the peer calculates friendships between its friends and the target peer contained in the message. To add more credit to the friends who share common ratings with the target peer, we use an

adjusted Pearson correlation method as a friendship measure. If the friendship measure is larger than the threshold, the friend ID with its rating information is sent back to the target peer. We call the users (or peers) suggested by friends, "potential friends". After the target peer receives reply messages from its friends, its friends' friends and so on, 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 for peers to adapt to dynamic changes and incrementally refine friend selection.

2.2 Scalability and Sparsity Problem

There are two kinds of CF algorithms: memory-based and model-based [4]. We took a memory-based approach in this paper because each peer keeps rating information of only its friends and itself and thus the data are not enough to generate an accurate model. Generally, memory-based approaches suffer from the scalability problem. However, for our approach, the problem is not severe because we keep only rating information for a moderate number of users.

The sparsity problem occurs when available data are insufficient for identifying similar users (neighbors). It is a major issue that limits the quality of recommendations and the applicability of CF in general. To date, several approaches [4] have been proposed to alleviate the sparsity problem under the environment that all rating information are kept in central hosts.

In our approach, the sparsity problem is not as severe as in the centralized approaches because the local database keeps only its friends' rating information. Our novel approach utilizes other agents' opinions (or predictions) instead of real ratings. Namely, an agent asks for friends' opinions when rating information in the local database is not sufficient. Some friends can give their opinion (not real rating just prediction) on an item based on their friends' ratings.

3. PERFORMANCE EVALUATION

We carried out experiments based on the Each-Movie data, which has been collected by Digital Equipment Research Center. To speed up our experiments, we only use a subset of the EachMovie data set. We use the MAE (Mean Absolute Error) as the accuracy metric. Prediction for an item is calculated by performing a weighted average of deviations from the neighbor's mean. We use the top *n* rule to select the nearest N neighbors based on the similarities of users.

Recommendation performance also depends on the number of friends. Fewer friends mean less information to use for recommendation. If the number of friends is over the optimum, it will add noise to reduce the recommendation performance. Based on our previous work [5], we set the friend number to 30 in our testing system.

The value of FDTL has a direct impact on recommendation performance. If there is no limitation of FDTL, the system will flood almost all of the nodes in the first run. In this case, FDTL functions as the TTL and causes a rush hour in the traffic. If the FDTL is too small, it cannot find sufficient resources to update the information. From our earlier experiments, we found that the system shows a favorable performance when FDTL is set to 4.

To simulate the initial environment, we randomly select 1000 users who have at least 20 ratings for items from Each-Movie data; we calculate the similarity among them and save 30 nearest friends for everyone. Then we randomly select 0 to 10 friends for each user.

We first randomly select 100 users and let them send out request packages to their friends if they have any. Those nodes are treated as the active nodes which spread the activity over the 1000 users and push them to find their top 30 friends. As shown in Figure 1, the recommendation performance increased quickly at the beginning runs, and then gradually increased until it tends to be flat after 25 runs. If most of users in the friend network cannot update their friend list any more based on current situation, we call this status "saturated". It is the optimum stage of the network. At the tail of runs, it reaches the saturated point for those 1000 users.

We also found the top 30 friends for those 1000 users by the central CF algorithm. The precision of distributed CF at saturated point is quite close to the centralized CF; their MAE values are 1.01 and 0.992, respectively. The coverage of distributed CF (97.4%) is greater than the central CF (73.2%) due to the ability of the distributed CF algorithm to address the sparsity problem.

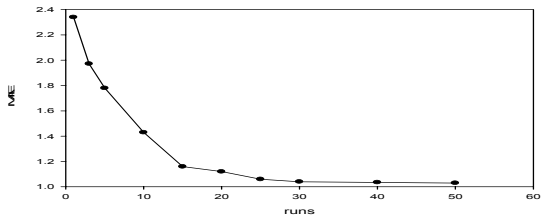

Figure 1. Recommendation Performance over Runs

4. CONCLUSIONS

We proposed a distributed CF algorithm based on a friend network, where users' rating information spreads out to their friends by a self-organizing protocol. From our experiments, we found that the accuracy of our approach is almost the same as a centralized CF approach while dealing well with the scalability problem and the sparsity problem. In the future, intensive experiments are needed to analyze the characteristics of our approach.

5. ACKNOWLEDGMENTS

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

6. REFERENCES

[1] K. Ali and V.-S. Wijnand, TiVo: Making Show Recommendations Using a Distributed Collaborative Filtering Architecture, In Proc. of KDD'04, 2004.

[2] Y. Chawathe, S. Ratnasamy and L. Breslau, Making Gnutella-like P2P Systems Scalable, In Proc. of ACM SIGCOMM'03, 2003.

[3] P. Han, B. Xie, F. Yang and R. Shen, A Novel Distributed Collaborative Filtering Algorithm and Its Implementation on P2P Overlay Network, In Proc. of PAKDD 2004, 2004.

[4] H. Hofmann, Latent Semantic Models for Collaborative Filtering, ACM Transactions on Information Systems, Vol.22, No.1, 2004.

[5] B. M. Kim, Q. Li, C. S. Park and S. Kim, A New Approach for Combining Content-based and Collaborative Filters, Journal of Intelligent Information Systems, 2006 (In Press).

[6] C. Pitsilis and L. Marshall, A Proposal for Trust-enabled P2P Recommendation Systems, Technical Report Series (CS-TR-910), University of Newcastle upon Tyne, 2005.