Overlay Management for Fully Distributed User-Based Collaborative Filtering*

Róbert Ormándi[1], István Hegedűs[1], and Márk Jelasity[2]

[1] University of Szeged, Hungary
{ormandi,ihegedus}@inf.u-szeged.hu
[2] University of Szeged and Hungarian Academy of Sciences, Hungary
jelasity@inf.u-szeged.hu

Abstract. Offering personalized recommendation as a service in fully distributed applications such as file-sharing, distributed search, social networking, P2P television, etc, is an increasingly important problem. In such networked environments recommender algorithms should meet the same performance and reliability requirements as in centralized services. To achieve this is a challenge because a large amount of distributed data needs to be managed, and at the same time additional constraints need to be taken into account such as balancing resource usage over the network. In this paper we focus on a common component of many fully distributed recommender systems, namely the overlay network. We point out that the overlay topologies that are typically defined by node similarity have highly unbalanced degree distributions in a wide range of available benchmark datasets: a fact that has important—but so far largely overlooked—consequences on the load balancing of overlay protocols. We propose algorithms with a favorable convergence speed and prediction accuracy that also take load balancing into account. We perform extensive simulation experiments with the proposed algorithms, and compare them with known algorithms from related work on well-known benchmark datasets.

1 Introduction

Offering useful recommendations to users of fully distributed systems is clearly a desirable function in many application domains. Some examples for larger efforts towards this goal are the Tribler platform [1] and more recently the Gossple project [2]. A fully distributed approach is also more preferable relative to centralized solutions, due to the increasing concerns over privacy.

However, the problem is also extremely challenging. Apart from the fact that centralized recommender systems—although working reasonably sometimes—are still far from perfect, offering good recommendations in fully distributed systems involves a number of special problems like efficiency, security and reliability, to name just a few.

In this work we focus on a class of recommender systems, the so called user-based collaborative filtering algorithms that are fairly simple, yet provide a reasonable performance [3]. The key concept is a similarity metric over the users, and recommendations are made on the basis of information about similar users.

* M. Jelasity was supported by the Bolyai Scholarship of the Hungarian Academy of Sciences. This work was partially supported by the Future and Emerging Technologies programme FP7-COSI-ICT of the European Commission through project QLectives (grant no.: 231200).

P. D'Ambra, M. Guarracino, and D. Talia (Eds.): Euro-Par 2010, Part I, LNCS 6271, pp. 446–457, 2010.

This idea also naturally lends itself to a distributed implementation, as it can be easily supported by similarity-based overlay networks as a simple service, that also have applications in other domains such as search. Indeed, many distributed protocols from related work follow this path in some way or another.

In this work we would like to shed light on the effects of the basic design choices in this domain with respect to recommendation performance, convergence time, and the balancing of the network load that the system generates during its operation.

Our contribution is threefold. First, we draw attention to the potential load balancing problem in distributed systems that manage similarity-based overlays for any purpose including recommendation or search. Second, we propose novel algorithms for similarity-based overlay construction. Third, we perform extensive simulation experiments on large benchmark datasets and compare our set of algorithms with each other and with a number of baselines. We measure prediction performance, examine its convergence and dynamics, and we measure load balancing as well.

2 Related Work

First we overview relevant ideas in recommender systems in general, and subsequently we discuss related work in the fully distributed implementations of these ideas, as well as additional related work that are based on similar abstractions.

A recommender system can be viewed as a service which supports e-commerce activities by providing items of interest for the users [4]. These algorithms are often centralized and Web-based operating on huge amounts of data—mainly on the previous ratings of the users. The algorithms which are based on the previous ratings of other *similar* users follow the so-called collaborative filtering (CF) approach. They are based on the simple heuristic that people who agreed (or disagreed) in the past will probably agree (or disagree) again. Thus, the predicted rate of an unseen item for a given user can be estimated on the basis of the rates of other users with *similar tastes*.

In the field of CF algorithms there exist numerous approaches. *User-based* approaches try to model the rating of a given item for a user by an aggregation of ratings of other users on the same item [3]. Although these approaches are very simple and intuitive, they provide a relatively good performance [5]. User-based CF algorithms are modular, hence they can be used with different aggregation methods and similarity metrics. One widely-used aggregation method is

$$\hat{r}_{u,i} = \frac{\sum_{v \in N_u} s_{u,v} \left(r_{v,i} - \bar{r}_v \right)}{\sum_{v \in N_u} |s_{u,v}|} + \bar{r}_u \tag{1}$$

defined in [6], where $r_{u,i}$ and $\hat{r}_{u,i}$ denote the known and the predicted rate of item i by user u, $\bar{r}_u$ and N_u denote the average rate and the neighbor set of user u, and $s_{u,v}$ measures the similarity between user u and v (e.g. Cosine similarity [3] or Pearson similarity [3] can be employed).

Our preliminary experiments showed that (among several variants) the aggregation method in (1) combined with the Cosine user similarity gives the best performance on our particular benchmarks. Since the focus of the present work is not recommendation performance per se, but the analysis of several distributed implementations of the basic idea of user-based CF, we fixed these methods in our experiments.

We should mention that there are numerous other approaches for recommendation such as the ones based on machine learning [7,8], matrix factorization [9], generative models [10], clustering [11,8], and dimension-reduction [7,12].

Moving on to distributed methods, we emphasize that we focus on P2P recommendation, and not on parallel implementations of centralized recommender techniques (such as matrix factorization, etc.). We consider only works that go beyond a simple idea and present at least some evaluations on benchmarks.

The largest group of methods define an overlay network based on some sort of similarity, and define a recommender algorithm on this network. For example, [4] and [13] follow this approach, although the overlay construction itself is not discussed or it is assumed to be done offline. The recommender algorithms then perform a search in this overlay up to a certain depth or up to a certain level of similarity, and aggregate the matching users with a standard method.

A slightly weaker approach is described in [14], where only a random network is assumed and the recommendation problem is treated as a search problem where a node needs to find similar users using a flooding based unstructured search.

A somewhat surprising result is described by Bakker at al [15], where they argue that in fact it is enough to take a random sample of the network and use the closest elements of that sample to make recommendations. Our results are consistent with this observation, although we describe better and equally cheap alternatives.

A more sophisticated approach is described by Bickson et al [16]. They define recommendation as a smoothing operation over a social network, which is expressed as a minimization problem using an objective function that expresses the requirements for the recommendation. The problem is solved by using an iterative method. Unfortunately no results are given on recommender system benchmarks due to the slightly different formulation of the basic problem.

It is of course possible to apply distributed hash tables [17]. Here, users are stored in a hash table and they are indexed by (item, rate) pairs as keys. Using this data structure, the users for a given item and rate are available from the distributed hash table (DHT) on demand. This method is not scalable if there are many recommendations to be made in the system, since the necessary information is not always available locally.

One of the most detailed studies on distributed recommender systems with performance evaluation can be found in [18]. The proposed models were implemented on the basis of the BUDDYCAST [19] overlay management service, which is the main overlay management method of the Tribler file sharing protocol [1]. We used our own implementation of this model as a baseline method, since the original study [18] did not carry out load balancing measurements.

Finally, although not directly related to the recommender systems, the area of exploiting semantic proximity for search also involves building overlay networks based on node similarity and therefore our algorithms and observations are relevant in this area as well. Examples of research in this area are described in [2,20,21,22].

3 Interesting Properties of CF Datasets

In our simulations we applied three different benchmark datasets, namely the MovieLens [5] dataset, the Jester [12] dataset and the BookCrossing [23] dataset. In this section we introduce these benchmarks and show some of their properties that raise interesting—and so far largely overlooked—problems in distributed environments.

Table 1. Basic statistics of datasets

	MovieLens	Jester	BookCrossing
# users	71,567	73,421	77,806
# items	10,681	100	185,974
size of train	9,301,274	3,695,834	397,011
sparsity	1.2168%	50.3376%	0.0027%
size of eval	698,780	440,526	36,660
eval/train	7.5127%	11.9195%	9.2340%
# items $\geq$	20	15	1
rate set	$1, \ldots, 5$	$-10, \ldots, 10$	$1, \ldots, 10$
MAE(med)	0.93948	4.52645	2.43277

Table 1 summarizes some basic statistics of our datasets. In the case of MovieLens we used the official r_a partition so that its evaluation set contained 10 ratings per user. For Jester and BookCrossing we produced the evaluation set as proposed in [15]: we withheld 6 ratings from the training set where possible (if the user under consideration had at least 6 rated items). In this table '# items $\geq$' means the minimal number of items rated by some user. Sparsity denotes the ratio of existing and possible rates in the training sets. The value MAE(med) is a trivial baseline for prediction performance; it is defined as the mean absolute error (MAE) computed on the evaluation set using the median-rate of training set as a prediction value. Clearly, a very significant difference can be found in properties related to sparsity. This will have significant implications on the performance of our algorithms, as we show later.

As mentioned before, in distributed settings one suitable and popular approach is to build and manage an overlay that connects similar users. This overlay can be viewed as a graph where each node corresponds to a user and there is a directed edge between user A and B if and only if user B belongs to the most similar users of A. This overlay plays an important role in a P2P recommender system. First, the performance of the recommendation depends on the structure of the overlay. Second, the costs and load balancing of the overlay management protocol depend on the topology of this similarity network.

To the best of our knowledge, the second role of the similarity overlay has not been addressed so far in the literature. Nevertheless it is an important issue, since the load generated by the overlay management process might correlate with the number of nodes that link to a given node as one of its most similar nodes. More precisely, the load of a node might correlate with its in-degree in the overlay network. Thus, if the in-degree distribution of the overlay network is extremely unbalanced (e.g. if it has a power-law distribution), some of the nodes can experience a load that is orders of magnitudes higher than the average. Thus, it is very important to consider the in-degree distribution of the overlay when planning a P2P recommender system, and examine the incurred loads on the individual nodes as a function of this distribution.

Figure 1 shows the in-degree distributions of the k nearest neighbor (kNN) overlay of each benchmark dataset. In this overlay each node has k directed outgoing edges to the k most similar nodes. As can be seen from the plots, the BookCrossing dataset has an almost power-law in-degree distribution, with many nodes having incoming links from almost every other node (note that the size of this dataset is around 77,806 users).

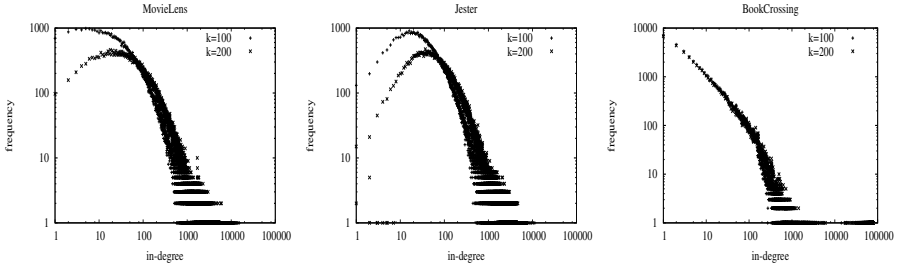

Fig. 1. In-Degree Distribution of Benchmark Datasets

To see whether this might be a general property of high dimensional datasets, we need to consider some basic properties of high dimensional metric spaces. If we generate high dimensional uniform random datasets from the unit cube and construct their kNN graphs, we will find that most of the points lie on the convex hull of the dataset. These points are mostly situated at the same distance from each other. The nodes corresponding to these points have a mostly uniform and relatively small in-degree in the kNN graph. The very few points inside the convex hull are close to a huge number of points on the convex hull, and so have high in-degree.

These observations indicate that we have to explicitly take into account load balancing when building a recommender system in a fully distributed manner.

4 Algorithms

The algorithms we examine all rely on building and managing a user-similarity overlay. In the top level of the protocol hierarchy, they apply the same user-based CF algorithm for making recommendations, strictly using locally available information (that is, information about the neighbors in the overlay).

Since we focus on overlay management, we fix the recommender algorithm and not discuss it any further. As it was mentioned in the previous sections, for this we need an aggregation method and a user similarity metric. We selected the aggregation shown in (1), proposed in [6]. Our similarity metric is Cosine similarity, which achieved the best performance on our benchmarks. Note that the selected user similarity is of course known to the overlay management algorithm and is used to direct the overlay construction.

We also assume that the local views of the nodes contain not only the addresses of the neighbors, but also a descriptor for each neighbor, that contains ratings made by the corresponding user. This implies that computing recommendation scores do not load the network since all the necessary information is available locally. However, there is a drawback; namely the stored information is not up-to-date. As we will show later, this is not a serious problem since on the one hand, recommendation datasets are not extremely dynamic and, on the other hand, the descriptors are in fact refreshed rather frequently due to the management algorithms.

In sum, the task of overlay management is to build and maintain the best possible overlay for computing recommendation scores, by taking into account bandwidth usage

Algorithm 1. Random Nodes based Overlay Management

Parameters: k: the size of view; r: the number of randomly generated nodes

```
1. while true do
2.     samples ← getRandomPeers(r)
3.     for i = 1 to r do
4.         peer ← get(samples, i)
5.         peerDescriptor ← descriptor(peer)
6.         insert(view, peerDescriptor)
```

at the nodes. We expect a minimal, uniform load from overlay management even when the in-degree distribution of the expected overlay graph is unbalanced.

4.1 BUDDYCAST Based Recommendation

As we mentioned earlier we applied the BUDDYCAST overlay management protocol as a baseline method. Now we give a very brief overview of this algorithm and its numerous parameters; for details please see [19].

The algorithm maintains a number of lists containing node descriptors. The taste buddy list contains the most similar users (peers), all those who communicated with the node before. The recommendation for a peer is calculated based on this list.

The BUDDYCAST algorithm contains a mechanism for load balancing: a block list. Communication with a peer on the block list is not allowed. If a node communicates with another peer, it is put on the block list for four hours.

Finally, a node also maintains a candidate list, which contains close peers for potential communication, as well as a random list that contains random samples from the network. For overlay maintenance, each node periodically (in every 15 seconds by default) connects to the best node from the candidate list with probability α, and to a random list with probability $1 - \alpha$, and exchanges its buddy list with the selected peer.

4.2 kNN Graph from Random Samples

We assume that a node has a local view of size k that contains node descriptors. These will be used by the recommender algorithm.

In Algorithm 1 each node is initialized with k random samples from the network, and they iteratively approximate the kNN graph. The convergence is based on a random sampling process which generates r random nodes from the whole network in each iteration. These nodes are inserted into the view which is implemented as a bounded priority queue. The size of this queue is k and the priority is based on the similarity function provided by the recommender module.

Applying a priority queue here on the basis of similarities means that nodes remember the most similar nodes from the past iterations. This means that since random samples are taken from the entire network, each node will converge to its kNN view with positive probability.

Method GETRANDOMPEERS can be implemented, for example, using the NEWSCAST [24] protocol.

This algorithm does converge, as argued above, albeit very slowly. However, it is guaranteed to generate an almost completely uniform load since the only communication that takes place is performed by the underlying peer sampling implementation (NEWSCAST), which has this property.

4.3 kNN Graph by T-MAN

We can manage the overlay with the T-MAN algorithm as well [25]. This algorithm manages a view of size k, as in the random algorithm above. T-MAN periodically updates this view by first selecting a peer node to communicate with, then exchanging its view with the peer, and finally merging the two views and keeping the closest k descriptors. This is very similar to Algorithm 1, but instead of r random samples the update is performed using the k elements of the view of the selected peer.

In this paper we examine the following methods for T-MAN which are employed as peer selection methods:

Global: This approach selects the node for communication from the whole network randomly. This can be done by using a NEWSCAST layer as it was described in the previous section. We expect this approach to distribute the load in the network uniformly since with this selection the incoming communication requests do not depend on the in-degree of the kNN graph at all.

View: In this approach the node for communication is selected from the view of the current node uniformly at random. The mechanism of this selection strategy is similar to the previous one, but the spectrum of the random selection is smaller since it is restricted to the view instead of the whole network.

Proportional: This approach also selects a node for view exchange from the view of the current node, but here we define a different probability distribution. This distribution is different for each node and it is reversely proportional to the value of a selection counter, which measures the load of the node in the previous time interval. The exact definition of the selection probability for a neighbor j of node i is

$$p_{i,j} = \frac{\frac{1}{sel_j+1}}{\sum_{k \in View_i} \frac{1}{sel_k+1}}, \tag{2}$$

where sel_k is the value of the selection counter of the kth neighbor. This information is stored in the node descriptors. The motivation for this selection method is to reduce the load on the nodes that have a high in-degree in the kNN graph, while maintaining the favorable convergence speed of the T-MAN algorithm.

Best: The strategy that selects the most similar node for communication without any restriction. We expect that this strategy converges the most aggressively to the perfect kNN graph, but at the same time it results in the most unbalanced load.

4.4 Randomness Is Sometimes Better

Our experimental results (to be presented in Section 6) indicated that in certain cases it is actually *not* optimal to use the kNN view for recommendation. It appears to be the case that a more relaxed view can give better recommendation performance.

To test this hypothesis, we designed a randomization technique that is compatible with any of the algorithms above. The basic idea is that we introduce an additional parameter, $n \leq k$. The nodes still have a view of size k, and we still use the same recommender algorithm based on these k neighbors. However, we apply any of the algorithms above to construct a (k-n)NN overlay graph (not a kNN graph), and we fill the remaining n elements in the following way: we take $r \geq n$ random samples (not necessarily independent in each cycle) and we take the closest n nodes from this list. With $n = k$ we get the algorithms proposed in [15], and with $n = 0$ this modification has no effect, so we get the original algorithm for constructing the kNN graph.

5 System Model

We consider a set of nodes connected through a routed network. Each node has an address that is necessary and sufficient for sending a message to it. To actually communicate, a node has to know the address of the other node. This is achieved by maintaining a *partial view* (*view* for short) at each node that contains a set of node descriptors. Views can be interpreted as sets of edges between nodes, naturally defining a directed graph over the nodes that determines the topology of an *overlay network*.

Although the class of algorithms we discuss has been shown to tolerate unpredictable message delays and node failures well [25,24], in this work we focus on load balancing and prediction performance, so we assume that messages are delivered reliably and without delay, and we assume that the nodes are stable.

Finally, we assume that all nodes have access to the peer sampling service [24] that returns random samples from the set of nodes in question. We will assume that these samples are indeed random. The results presented in [24] indicate that the peer sampling service has realistic implementations that provide high quality samples at a low cost.

6 Empirical Results

We implemented our protocols and performed our experiments in PeerSim [26,27]. We performed a set of simulations of our algorithms with the following parameter value combinations: *view update* is random or T-MAN; *peer selection* for T-MAN is GLOBAL, VIEW, BEST or PROPORTIONAL; and the number of *random samples* is 20, 50, or 100 for random, and 0 or 100 for T-MAN.

The BUDDYCAST algorithm was implemented and executed with the following parameters: the size of the buddy list and the candidate list was 100, the size of the random list was 10, and α was 0.5. The size of the block list had to be restricted to be 100 as well, in order to be able to run our large scale simulations. The view size for the rest of the protocols was fixed at $k = 100$ in all experiments for practical reasons: this represents a tradeoff between a reasonably large k and the feasibility of large scale simulation.

In these simulations we observe the prediction performance in terms of the MAE measure and the distribution of the number of incoming messages per cycle at a node. Note that the number of outgoing messages is exactly one in each case.

Let us first discuss the effect of parameter r. This is a crucial parameter for random view update, while in the case of T-MAN the role of random samples is merely to help

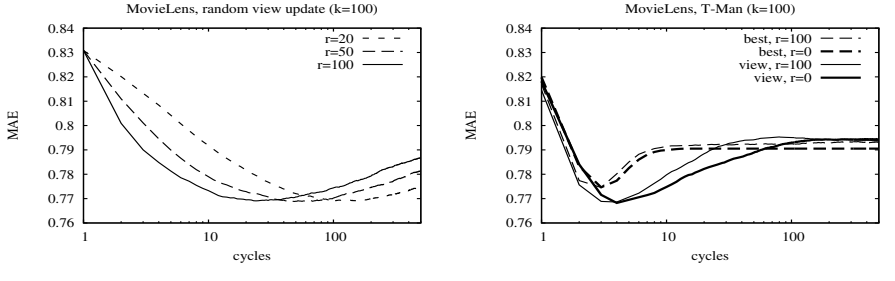

Fig. 2. Effect of parameter r in a few settings

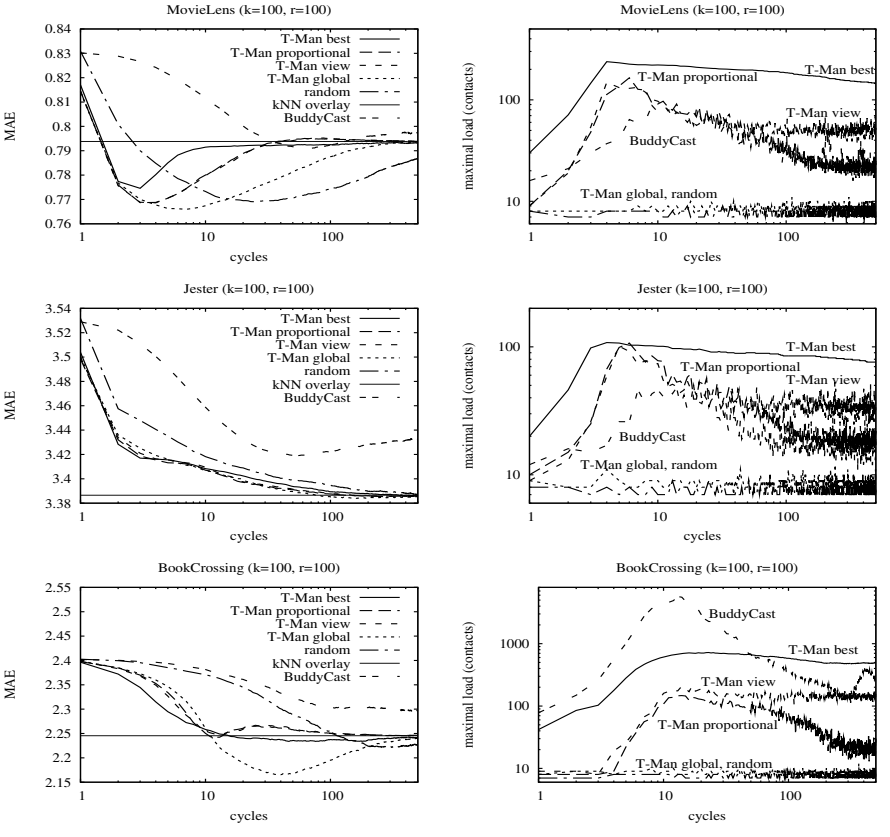

Fig. 3. Experimental results. The scale of the plots on the right is logarithmic.

the algorithm to avoid local optima, and to guarantee convergence. Figure 2 shows the effect of r in the case of the MovieLens database. The effect of r on the other databases and for other settings is similar.

We can observe that in the case of a random view update, r simply is a multiplicative factor that determines the speed of convergence: twice as many samples per cycle result in a halving of the necessary cycles to achieve the same value. In the case of T-MAN, the version with random samples converges faster, while the generated load remains the same (not shown). Accordingly, in the following we discuss T-MAN algorithms only with $r = 100$, and random view update algorithms only with $r = 100$.

In Figure 3 we show the results of the experiments, where the MAE and the maximal load is illustrated. The maximal load is defined as the maximal number of incoming messages any node receives during the given cycle. The first interesting observation is that the load balancing property of the different algorithms shows a similar pattern over the three datasets, however, the convergence of the MAE is rather different (see also Table 1). In particular, in the case of the MovieLens and BookCrossing benchmarks the MAE reaches a minimum, after which it approaches the top-k based prediction from below, whereas we do not see this behavior in the much denser Jester database.

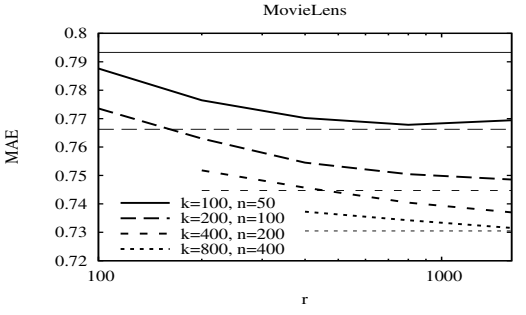

Fig. 4. Effect of adding randomness to the view. Thin horizontal lines show the $n = 0$ case

Indeed, the reason for this behavior lies in the fact that for the sparse datasets a larger k is a better choice, and our setting ($k = 100$) is actually far from optimal. In the initial cycles the view approximates a random sample from a larger k parameter. To verify this, we calculated the MAE of the predictions based on the algorithm described in Section 4.4. The results are shown in Figure 4 later on.

It is clear that for a small k it is actually better *not* to use the top k from the entire network; rather it is better to fill some of the views with the closest peers in a relatively small random sample from the network. Especially for the smallest k we examined ($k = 100$) this technique results in a significant improvement in the MAE compared to the recommendation based on the closest k peers in all datasets. This algorithm can easily be implemented, since we simply have to combine any of the convergent algorithms with an appropriate setting for k (such as $k = 50$) and use a peer sampling service to add to this list the best peers in a random sample of a given size.

As a closely related note, the random view update algorithms can be "frozen" in the state of minimal MAE easily, without any extra communication, provided we know in advance the location (that is, the cycle number) of the minimum. Let us assume it is in cycle c. Then we can use, for a prediction at any point in time, the best k peers out of the union of $c \cdot r$ random samples collected in the previous c cycles, which is very similar to the approach taken in [15].

Clearly, the fastest convergence is shown by the T-MAN variants, but these result in unbalanced load at the same time. The PROPORTIONAL variant discussed in Section 4.3 reduces the maximal load, however, only when the topology has already converged. During the convergence phase, PROPORTIONAL behaves exactly like the variant VIEW.

Quite surprisingly, the best compromise between speed and load balancing seems to be GLOBAL, where the peer is selected completely at random by T-MAN. In many topologies, such as a 2-dimensional grid, a random peer possesses no useful information for another node that is far from it in the topology, so we can in fact expect to do worse than the random view update algorithm. However, in target graphs such as kNN graphs based on similarity metrics, a large proportion of the network shares useful information, namely the addresses of the nodes that are more central.

On such unbalanced graphs T-MAN GLOBAL is favorable, because it offers a faster convergence than a pure random search (in fact, it converges almost as fast as the more aggressive T-MAN variants), however, the load it generates over the network is completely identical to that of random search, and therefore the maximal load is very small:

the maximum of N samples from a Poisson distribution with a mean of 1 (where N is the network size). In addition, the node with the maximal load is different in each cycle.

Finally, we can observe that on the BookCrossing database some algorithms, especially BuddyCast and T-MAN with BEST peer selection, result in an extremely unbalanced degree distribution (note the logarithmic scale of the plot). This correlates with the fact that the BookCrossing database has most unbalanced degree distribution (see Figure 1). Even though we have not optimized the parameters of BuddyCast, this result underlines our point that one has to pay attention to the in-degree distribution of the underlying kNN graph.

7 Conclusions

In this paper we tackled the problem of the construction of similarity-based overlay networks with user-based collaborative filtering as an application. We pointed out that similarity-based overlays can have a very unbalanced degree distribution, and this fact might have a severe impact on the load balancing of some overlay management protocols. The main conclusion that we can draw is that in highly unbalanced overlays (that are rather frequent among similarity-based networks) the overlay construction converges reasonably fast even in the case of random updates; or, with T-MAN, uniform random peer selection from the network. At the same time, the traditional, aggressive peer selection strategies that have been proposed by other authors should be avoided because they result in a highly unbalanced load experienced by the nodes.

In sum, in this domain T-MAN with global selection is a good choice, because it has a fully uniform load distribution combined with an acceptable convergence speed, which is better than that of the random view update. However, care should be taken because this conclusion holds only in these unbalanced domains, and in fact this algorithm is guaranteed to perform extremely badly in large-diameter topologies.

References

1. Garbacki, P., Iosup, A., Doumen, J., Roozenburg, J., Yuan, Y., Brinke, T.M., Musat, L., Zindel, F., van der Werf, F., Meulpolder, M., et al.: Tribler protocol specification
2. Kermarrec, A.M.: Challenges in personalizing and decentralizing the web: An overview of GOSSPLE. In: Guerraoui, R., Petit, F. (eds.) SSS 2009. LNCS, vol. 5873, pp. 1–16. Springer, Heidelberg (2009)
3. Adomavicius, G., Tuzhilin, E.: Toward the next generation of recommender systems: A survey of the state-of-the-art and possible extensions. IEEE Trans. on Knowledge and Data Engineering 17, 734–749 (2005)
4. Pitsilis, G., Marshall, L.: A trust-enabled P2P recommender system. In: Proc. 15th IEEE Intl. Workshops on Enabling Technologies: Infrastructure for Collaborative Enterprises (WETICE 2006), pp. 59–64 (2006)
5. Herlocker, J.L., Konstan, J.A., Borchers, A., Riedl, J.: An algorithmic framework for performing collaborative filtering. In: Proc. 22nd annual Intl. ACM SIGIR Conf. on Research and development in information retrieval (SIGIR 1999), pp. 230–237. ACM, New York (1999)
6. Resnick, P., Iacovou, N., Suchak, M., Bergstrom, P., Riedl, J.: Grouplens: an open architecture for collaborative filtering of netnews. In: Proc. 1994 ACM Conf. on Computer supported cooperative work (CSCW 1994), pp. 175–186. ACM, New York (1994)
7. Billsus, D., Pazzani, M.J.: Learning collaborative information filters. In: Proc. 15th Intl. Conf. on Machine Learning (ICML 1998), pp. 46–54. Morgan Kaufmann, San Francisco (1998)

8. Park, Y.-J., Tuzhilin, A.: The long tail of recommender systems and how to leverage it. In: Proc. 2008 ACM Conf. on Recommender systems (RecSys 2008), pp. 11–18. ACM, New York (2008)
9. Takács, G., Pilászy, I., Németh, B., Tikk, D.: Scalable collaborative filtering approaches for large recommender systems. Journal of Machine Learning Research 10, 623–656 (2009)
10. Lawrence, N.D., Urtasun, R.: Non-linear matrix factorization with gaussian processes. In: Proc. 26th Annual Intl. Conf. on Machine Learning (ICML 2009), pp. 601–608. ACM, New York (2009)
11. O'Connor, M., Herlocker, J.: Clustering items for collaborative filtering. In: Workshop on Recommender Systems at 22nd ACM SIGIR (1999)
12. Goldberg, K., Roeder, T., Gupta, D., Perkins, C.: Eigentaste: A constant time collaborative filtering algorithm. Information Retrieval 4(2), 133–151 (2001)
13. Castagnos, S., Boyer, A.: Modeling preferences in a distributed recommender system. In: Conati, C., McCoy, K., Paliouras, G. (eds.) UM 2007. LNCS (LNAI), vol. 4511, pp. 400–404. Springer, Heidelberg (2007)
14. Tveit, A.: Peer-to-peer based recommendations for mobile commerce. In: Proc. 1st Intl. workshop on Mobile commerce (WMC 2001), pp. 26–29. ACM, New York (2001)
15. Bakker, A., Ogston, E., van Steen, M.: Collaborative filtering using random neighbours in peer-to-peer networks. In: Proc. 1st ACM Intl. workshop on Complex networks meet information & knowledge management (CNIKM 2009), pp. 67–75. ACM, New York (2009)
16. Bickson, D., Malkhi, D., Zhou, L.: Peer-to-Peer rating. In: Proc. 7th IEEE Intl. Conf. on Peer-to-Peer Computing, 2007 (P2P 2007), pp. 211–218. IEEE Computer Society, Los Alamitos (2007)
17. Han, P., Xie, B., Yang, F., Shen, R.: A scalable P2P recommender system based on distributed collaborative filtering. Expert Systems with Applications 27(2), 203–210 (2004)
18. Wang, J., de Vries, A.P., Reinders, M.J.T.: Unified relevance models for rating prediction in collaborative filtering. ACM Trans. on Information Systems (TOIS) 26(3), 1–42 (2008)
19. Pouwelse, J., Yang, J., Meulpolder, M., Epema, D., Sips, H.: Buddycast: an operational peer-to-peer epidemic protocol stack. In: Proc. 14th Annual Conf. of the Advanced School for Computing and Imaging, ASCI, pp. 200–205 (2008)
20. Voulgaris, S., van Steen, M.: Epidemic-style management of semantic overlays for content-based searching. In: Cunha, J.C., Medeiros, P.D. (eds.) Euro-Par 2005. LNCS, vol. 3648, pp. 1143–1152. Springer, Heidelberg (2005)
21. Garbacki, P., Epema, D.H.J., van Steen, M.: A two-level semantic caching scheme for super-peer networks. In: Proc. 10th Intl. Workshop on Web Content Caching and Distribution (WCW 2005), pp. 47–55. IEEE Computer Society, Los Alamitos (2005)
22. Akavipat, R., Wu, L.S., Menczer, F., Maguitman, A.: Emerging semantic communities in peer web search. In: Proc. Intl. workshop on Information retrieval in peer-to-peer networks (P2PIR 2006), pp. 1–8. ACM, New York (2006)
23. Ziegler, C.N., McNee, S.M., Konstan, J.A., Lausen, G.: Improving recommendation lists through topic diversification. In: Proc. 14th Intl. Conf. on WWW, pp. 22–32. ACM, New York (2005)
24. Jelasity, M., Voulgaris, S., Guerraoui, R., Kermarrec, A.M., van Steen, M.: Gossip-based peer sampling. ACM Trans. on Computer Systems 25(3), 8 (2007)
25. Jelasity, M., Montresor, A., Babaoglu, O.: T-Man: Gossip-based fast overlay topology construction. Computer Networks 53(13), 2321–2339 (2009)
26. Montresor, A., Jelasity, M.: Peersim: A scalable P2P simulator. In: Proc. Ninth IEEE Intl. Conf. on Peer-to-Peer Computing (P2P 2009), pp. 99–100. IEEE, Los Alamitos (2009) (extended abstract)
27. Jelasity, M., Montresor, A., Jesi, G.P., Voulgaris, S.: The Peersim simulator, `http://peersim.sf.net`