Emergence of Spontaneous Order Through Neighborhood Formation in Peer-to-Peer Recommender Systems

Ernesto Díaz-Avilés Lars Schmidt-Thieme Cai-Nicolas Ziegler

Institut für Informatik, Universität Freiburg
Georges-Köhler-Allee, Gebäude 51
79110 Freiburg i.Br., Germany

{diaz,lst,cziegler}@informatik.uni-freiburg.de

ABSTRACT

The advent of the Semantic Web necessitates paradigm shifts away from centralized client/server architectures towards decentralization and peer-to-peer computation, making the existence of central authorities superfluous and even impossible. At the same time, recommender systems are gaining considerable impact in e-commerce, providing people with recommendations that are personalized and tailored to their very needs. These recommender systems have traditionally been deployed with stark centralized scenarios in mind, operating in closed communities detached from their host network's outer perimeter. We aim at marrying these two worlds, i.e., decentralized peer-to-peer computing and recommender systems, in one agent-based framework. Our architecture features an epidemic-style protocol maintaining neighborhoods of like-minded peers in a robust, self-organizing fashion. In order to demonstrate our architecture's ability to retain scalability, robustness *and* to allow for convergence towards high-quality recommendations, we conduct offline experiments on top of the popular MovieLens dataset.

Categories and Subject Descriptors

C.2.4 [**Computer-Communication Networks**]: Distributed Systems—*Distributed Applications*; H.3.3 [**Information Storage and Retrieval**]: Information Search and Retrieval—*Information Filtering*

General Terms

Algorithms, Experimentation, Performance

Keywords

Recommender systems, peer-to-peer networking, collaborative filtering, epidemic protocols

1. INTRODUCTION

Today's most successful recommender systems are part of e-commerce infrastructures, and they share a common characteristic: they are all based on a client/server architecture (or an N-tier variation of it), where the user profile information and recommendation engine are *centralized*.

However, the Semantic Web vision [4] that we share is more likely to be based on decentralized architectures, like the ones provided by peer-to-peer (P2P) overlay networks, where *agents* would interact via free information exchange or trading. We present an alternative to centralized collaborative filtering, exploiting the advantages of peer-to-peer networks.

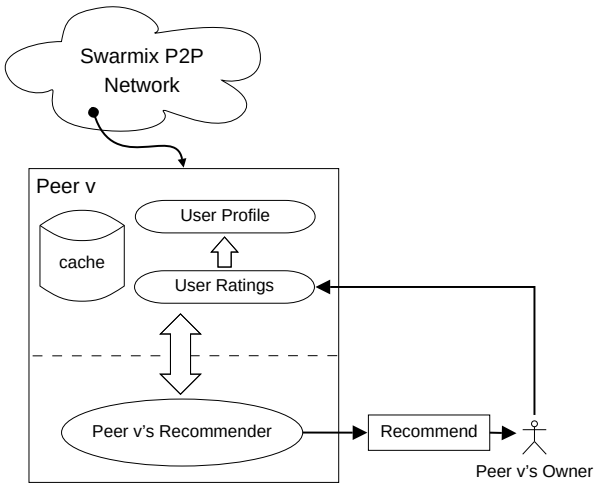

Figure 1: Overview of the Swarmix distributed recommender system architecture.

We introduce Swarmix, a distributed architecture (Figure 1) whose epidemic-style protocol is responsible for the overlay P2P network construction and maintenance. The protocol is able to associate each peer v with a fixed number of highly similar neighbors whose similarity with respect to v improves during the perpetual execution of the protocol. Each peer v runs a recommender system locally and is in control of its profile and ratings; v's recommendations are computed using only its peers; which requires no global knowledge of the network or access to a central server responsible for storing or computation.

WWW2005, May 10–14, 2005, Chiba, Japan.
.

The rest of the paper is organized as follows. In Section 2, we present a general model shared by epidemic-style protocols based on a push-pull mechanism. In Section 3, we introduce the Swarmix protocol at the core of our architecture. The distributed recommender system implementation is presented in Section 4. In Section 5, we present the experimental setup and evaluation metric used. In Section 6, we report our experimental results. In Section 7, we point to some related work. Finally, Section 8 presents our conclusions and future research.

2. THE GENERAL EPIDEMIC PUSH-PULL PROTOCOL (GEP3)

Epidemic-style distributed protocols like the *anti-entropy push-pull* mechanism [6], or the *Newscast* protocol [13, 14] share the same general model that we call the *general epidemic push-pull protocol* (GEP3), which is detailed in this section.

We formalize the task to distribute data in a peer-to-peer network as follows: Let V be a set representing the **peers**, S a set representing **data**. Then each peer $v \in V$ stores

1. the set of peers Q_v it is able to communicate with, called **neighborhood**[1],
2. a set $C_v \in \mathcal{P}(S)$ of data called the peers **cache**–where $\mathcal{P}(.)$ denotes the power set–, and
3. a **utility function** $u : \mathcal{P}(S) \to \mathbb{R}$ that specifies for each possible state of the cache the utility for the peer. For simplicity, we will only consider decomposable utility functions with $u(C) = \sum_{s \in C} u(s)$ in the following.

Initially, each peer may have some data, and new data or new versions of old data may enter the system through any peer, at any time. Data is transmitted through the network by exchanging and merging the caches of two neighboring peers v and w with the goal to maximize the utility of each peer's cache, conforming to some constraints as, e.g., a maximal cache size.

As several copies and versions of the same data may pile up during runtime at each peer, we need some method for duplicate elimination

$$C'_v := \text{merge}(C_v, C_w)$$

called **cache merging** in the following. For version handling, let $\text{id} : S \to \text{IDs}$ be a function that returns an ID for each data, and $t : S \to \text{Time}$ be a function that returns the *timestamp* of the data's version, then **merging by retaining the most recent version** for each data is implemented as follows:

$$\text{merge}(C_v, C_w) := \{\text{argmax}_{\substack{r \in C_v \cup C_w, \\ id(r)=\text{id}(s)}} t(r) \mid s \in C_v \cup C_w\}$$

Then, we **retain a fixed number of most useful data** using method

$$C''_v := \text{select}(C'_v, k)$$

called **selection function**, implemented as follows:

[1]The set V of peers and the neighborhoods $(Q_v)_{v \in V}$ form a directed graph called **network topology.**

Algorithm 1 General epidemic push-pull protocol (GEP3)

do forever

1. $w := \text{getRandomPeer}(Q_v)$;
2. **push** C_v to w;
3. **pull** C_w from w;
4. $C'_v := \text{merge}(C_v, C_w)$;
5. $C''_v := \text{select}(C'_v, k)$;
6. $Q'_v := \text{neighbors}(Q_v, C''_v)$;

$$\text{select}(C'_v) := \text{argmax}^k_{s \in C'_v} u(s)$$

where k is a fixed integer and argmax^k returns the k first elements in descending order of its argument.

Finally, the protocol allows to adapt the network topology dynamically based on local information, accomplished by a function

$$Q'_v := \text{neighbors}(Q_v, C''_v)$$

called **neighborhood adaptation**. If each data contains information about the peer at which it entered the network, modeled by a function $\text{src} : S \to V$, **neighborhood adaptation by retaining a fixed number of most useful peers** is implemented as

$$\text{neighbors}(Q_v, C''_v) := \text{src}(\text{argmax}^l_{s \in C''_v} u(s))$$

where l is a fixed integer. If **no neighborhood adaptation** should be used, i.e., neighborhoods remain constant during runtime, then $\text{neighbors}(Q_v, C''_v) := Q_v$.

Note that if each data item (and all its versions) can enter the network only at a single peer, and if there is exactly one such data item per peer, then the data ID coincides with the data source (id = src).

The **general epidemic push-pull protocol (GEP3)** ran by each peer $v \in V$ is outlined in Algorithm 1. Once each $v \in V$ has performed one run of Algorithm 1, a *cycle* has been completed.

Steps 4 to 6 are executed separately for peers v and w and in general may give different results, e.g., due to different utility functions used for selecting the most useful data. We say the GEP3 is **symmetric** if steps 4 to 6 are guaranteed to return the same results for both v and w.

Note that the three main steps, merging, selection, and neighborhood adaptation, are most commonly implemented using some sort of weight function that defines an ordering on the data or peers, but these weight functions do not have to be the same. Especially if versions are present, it makes more sense to use the timestamps' 'freshness' for merging, but the utility for selection and neighborhood adaption.

In the rest of this section, we briefly explore the matching characteristics between the anti-entropy push-pull mechanism, the Newscast protocol, and the general protocol presented above.

Anti-entropy push-pull [6]: this mechanism is targeted to replicate databases across a network. In this instance of the GEP3 the information collection $S_v = C_v$ of every $v \in V$ corresponds to database entries. Every single node v in this

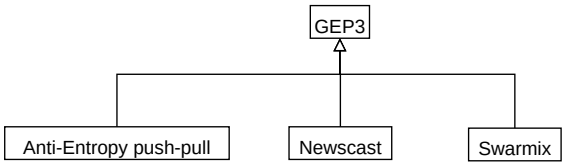

Figure 2: The GEP3 instances.

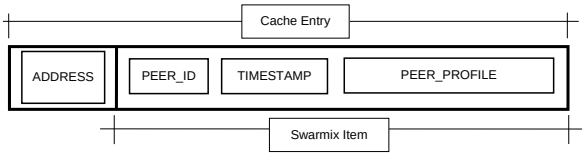

Figure 3: The format of Swarmix items and cache entries.

setup has a global view of the network, i.e., $Q_v = V$, and the goal is to always maintain *all* 'freshest' entries. Therefore, the merging and selection steps are both based on the function $t : S \rightarrow \text{Time}$, which associates timestamp values with the database entries.
After the nodes v and w have performed one run of Algorithm 1, both of them end with a *symmetric* information set, corresponding to all fresh database entries $C_v = C_w$. Note that this instance of the GEP3 requires no neighborhood adaptation.

Newscast protocol [14, 13]: in this case, news items defined by the protocol are the ones corresponding to the cache C. In contrast to the push-pull mechanism, the peers in the Newscast protocol have only a partial view of the system. The size of this network view (or neighborhood) is a fixed number k, which also corresponds to the size of the cache. This is a consequence of the specification of the news items, which includes information about the neighbors' IDs. Therefore, the partial view information is also subject to exchange when executing steps 2 and 3 of the GEP3 (i.e., the epidemic dissemination).

Similar to the push-pull mechanism, the weight function for merging and selection is also $t : S \rightarrow \text{Time}$, but a global synchronization is not required and just consistency between the timestamps inside a particular node collection is necessary. The selection function in this case keeps only the freshest k news items in the cache.

When a node v following the Newscast protocol updates not only its application-dependent information, but also its neighborhood, then every cycle it gets to know more and more nodes from which v selects the ones carrying the 'freshest' information.

After a run of the protocol, nodes v and w both end up with an information set corresponding to the freshest entries in their caches $C_v = C_w$, similar to the push-pull mechanism.

3. THE SWARMIX EPIDEMIC PROTOCOL

Our Swarmix epidemic protocol is based on the framework provided by the GEP3. In this section we detail the extensions of the model, and its specific characteristics (Figure 1).

Our protocol supposes a collection of *Swarmix items* called *cache*, corresponding to the set C_v in the GEP3. Each peer v in the overlay network locally stores its own cache, which has a limited number $k \in \mathbb{N}$ of *cache entries*.

The format of a Swarmix item stored at peer v is depicted in Figure 3. Suppose that the information contained in v's Swarmix item belongs to its neighbor w. This information includes w's unique ID (e.g., its pseudonym), a vector representation of its profile (e.g., its ratings), and a timestamp corresponding to the Swarmix item creation event. Note that, as in Newscast, no global synchronization is needed, just consistency between the timestamps inside a particular peer cache is required.

Note that k also corresponds to v's neighborhood size, i.e., to the size of set Q_v specified by the GEP3, which corresponds to v's "view" of the network. Obviously, this is a partial view, but a dynamic one, constantly changing as the protocol is executed.

The Swarmix item is also associated to w's network address, which is used by v to contact w when performing the protocol. This extra information, plus the Swarmix item itself, constitutes what we called a *cache entry*.

Note that the partial network view Q_v of peer v is also subject to information exchange when executing the push-pull steps, which in terms of GEP3, implies that the data ID coincides with the data source (i.e., id = src).

The GEP3 utility function u is defined to be the rating profile similarity function:

$$u_v(s) := \text{sim}(v, \text{id}(s))$$

which computes the similarity between peers v and w based on their respective rating profiles, e.g., cosine similarity.

The utility function assigns the similarity weight to each Swarmix item s in v's cache C_v, and permits to establish a total order over v's cache entries (i.e., $>_{s \in C_v}$), which constitutes the selection criteria to decide what Swarmix items are to be kept as the "best" ones, when performing the selection step of the GEP3.

We merge peers' v and w caches by retaining the most useful version of each Swarmix item:

$$\text{merge}(C_v, C_w) := \{\text{argmax}_{\substack{r \in C_v \cup C_w, \\ id(r)=\text{id}(s)}} u(r) \mid s \in C_v \cup C_w\}$$

As selection function as well as for neighborhood selection we opted for retaining a fixed number k of most useful peers as described above already. As the size of the cache and the size of the neighborhood are the same, the neighbors are just the peers specified by the Swarmix items in the updated cache after one round of the protocol.

Observations

1. The entries in v's and w's cache after one execution of the protocol are in general not the same (i.e., the Swarmix protocol is **asymmetric**), different to both of the protocols discussed before, i.e., anti-entropy push-pull and Newscast. Note that the asymmetry of caches and neighborhoods is not simply a consequence of an asymmetric similarity measure. In fact, cosine similarity itself is symmetric.

Table 1: Specific instances of the GEP3

GEP3			Anti-entropy push-pull	Newscast	Swarmix
C_v	,	cache at node v	database entries [6]	*news items* [14]	Swarmix items
Q_v	,	node v's view of the system (i.e. v's neighborhood)	global view ($Q_v = V$)	partial view of size k	partial view of size k
$u(s)$; $s \in C_v$	,	utility function	$t : S \rightarrow$ Time (i.e. timestamp)	$t : S \rightarrow$ Time (i.e. timestamp)	similarity(v, w) ; $v, w \in V$

2. The global communication cost of one cycle for the overall system depends on the cache size k, and the size of the user rating profile. In a cycle, each peer initiates exactly one information exchange session which involves the transfer of at most $2k$ cache entries. The size of the cache can be seen from Figure 3. Clearly, the global communication costs of one cycle grow linearly with the network size, but are evenly distributed over the set of peers.
   For the communication costs for a single peer, Jelasity et al. [14] prove that the distribution of the incoming connections is close to a Poisson distribution with $\lambda = 1$. Their proof is based on the random pickup step of the their Newscast epidemic algorithm, assuming unbiased random cache content, and for a large network size. Our protocol shares the same basic random pickup step, and under the same assumptions, we can expect that the same distribution of incoming connections applies to it.

3. From the point of view of *scalability* the sharing of communication cost over time, and among all peers is an important advantage: independent of the system size, each peer will experience the same predictable load without peaks.

4. Peers joining the network do not require any special sequence of communications, the new peer simply has to initialize its cache with at least one known peer (e.g., a *buddy*) which is already on-line, and start to execute the protocol.

5. A peer voluntarily leaving the overlay simply has to stop communicating, i.e., leavings are treated as failures.

4. RECOMMENDATION ALGORITHM

The problem space of automated collaborative filtering can be formulated as a matrix R of users versus items. Each cell of the matrix R represents a user's *rating* on a specific item, and each row corresponds to a *user profile*. The task of the recommender, under this formulation, is to predict values for specific, empty cells; i.e., to predict a user's rating for a not-yet-rated item.

A neighborhood-based collaborative filtering recommender system comprises the three fundamental steps described by Herlocker et al. [12]:

1. **Similarity computation**. Compute similarity of all users' profiles with the profile of the target user (i.e., the one requesting recommendations, also called active user).

2. **Neighborhood Formation**. Select a subset of users as a set of predictors: These neighbors correspond to the c most similar users for the active user.

3. **Aggregation and prediction computation**. The active user's profiles are aggregated computing the union of consumed items. The system also removes items already consumed by the active user, in order to guarantee that just new items are recommended. A weight is associated to each item based on its importance in the aggregation; consequently, the best N items, having the highest weights, are reported to the active user as the final recommendations.

These steps may overlap or might be slightly different depending on the specific recommender system that implements the algorithm.

In case of a recommender system operating in a standard client/server architecture, these steps are performed by the server in a centralized manner, while the role of the clients is minimal and limited to providing access to the final recommendation lists.

The Swarmix architecture distributes among all participants the execution of these three tasks. The neighborhood formation is provided by the Swarmix epidemic protocol, where every peer is associated with a neighborhood of like-minded peers.

This neighborhood corresponds to the partial view of the network (i.e., the cache) as discussed in the previous section. While the protocol executes, peer v's neighborhood changes continuously, selecting in every run the k most similar peers.

The similarity computation is also part of our epidemic protocol, and is performed by each peer in the overlay network. We perform the similarity computation using the *cosine-based* measure:

$$\text{sim}(v, w) := \cos(v, w) := \frac{\langle v, w \rangle}{||v|| \cdot ||w||}$$

where by abuse of notation v and w denote the respective rating profiles of peers v and w. Alternatively, any other similarity measure proposed in the literature could be used, e.g., Pearson correlation, Spearman rank, etc.

Finally, each peer is able to compute its recommendation list based on its neighborhood, that is, through its cache entries. For our architecture, we have implemented the *most-frequent items* approach suggested by Sarwar et al. [18].

Their technique can be seen as a majority voting election scheme, were each of the members of peer v's neighborhood casts a vote for each of the items he has consumed. Those N items with most votes, and new to the active user, are the recommendations.

5. EXPERIMENT OUTLINE

To evaluate the result of the top-N (with N=10) recommendations provided by our distributed architecture, we split the dataset into *training* and *test* set by randomly selecting a single rating (a *hidden* item) for each user to be part of the test set, and used the remaining ratings for training. Breese et al. [5] called this kind of experimental setup *all-but-1* protocol.

The nearest neighbors and top-10 recommendations were computed using the training set only.

The quality was measured by looking at the number of *hits*, which corresponds to the number of items in the test set that were also present in the top-N recommended items returned for each peer. More formally, *hit-rate*, is defined as

$$\text{hit-rate} := \frac{\sum_{v \in V} hit(v)}{|V|}$$

where *hit(v)* is a binary function that returns 1, if the hidden item is present in v's top-N list of recommendations, and 0 otherwise. A *hit-rate* value of 1.0 indicates that the system was able to always recommend the hidden item, whereas a *hit-rate* of 0.0 indicates that the system was not able to recommend any of the hidden items.

In order to ensure that our results are not sensitive to the particular training-test partitioning of each dataset, we performed five different runs for each of the experiments, each time using a different random partitioning into training and test sets. The results reported in the rest of this section are the averages over these five trials. Furthermore, to better compare the results we also present the confidence intervals for the mean estimation at a 95% confidence coefficient, when appropriate.

For the time analysis, 100 simulation cycles were performed. In each cycle, every peer v initiates a Swarmix communication session and, immediately after finishing, v's recommendations are computed. Thus, in each simulation cycle n Swarmix sessions are performed, and each peer v receives its top-10 recommendations once during the cycle.

In all experiments, a *cache* value of size 20 was used when executing the Swarmix protocol. This corresponds to the typical neighborhood size adopted for generic recommender systems.

The *bootstrapping procedure* for the P2P network is as follows. We start with a connected network, but with an initially unbalanced neighborhood structure with an average neighborhood size of 2.494 neighbors per peer. Therefore, none of the peers has its cache filled. This network state corresponds to simulation cycle 0, i.e., before starting to execute the Swarmix protocol.

The recommender system was also deployed in a centralized architecture, as a reference for the performance of our distributed architecture.

6. EXPERIMENTAL RESULTS

We evaluate the performance of the recommender system implemented using a 88% subset of the original 'small' MovieLens dataset[10] containing 943 users which have issued 88,263 explicit ratings, on a scale from 1 to 5, for 1,457 movies.[2] Each of the users has issued at least 15, and on average 93.6 ratings.

A user in the dataset becomes a *peer* in our simulation environment.

6.1 Neighborhood Quality

To evaluate the quality of the neighborhoods formed and maintained by the Swarmix epidemic protocol, we focused our attention on the average similarity score between each peer and its neighbors. For each of the 100 simulation cycles, Figure 4(a) shows the similarity values averaged over all peers in the overlay. The figure also presents, as reference, the average similarity values in the case of a centralized architecture, where each peer v is associated with its 20 most similar neighbors. Obviously, the similarity values for the centralized architecture remain constant throughout the simulation cycles, and they represent an upper bound for the performance of our distributed architecture.

The Swarmix protocol is able to maintain neighborhoods whose average similarity improves on every cycle, converging to values close to the ones in a centralized architecture. Furthermore, the variance of the average similarity scores decreases over time (Figure 4(b)). Note that the initial behavior is explained by the bootstrapping procedure used, which initializes a peer's cache not at full capacity, but with just 2 or 3 entries maximum. This makes the variance worse during the initial cycles, while filling the caches, but then starts improving to smaller values throughout the rest of the simulation.

Note that even though we have applied *significance weighting* to improve recommendation quality, as recommended by Herlocker et al. [12], the similarity values reported here correspond to the ones *before* the application of such weighting. The reason is that we want to first assess the performance of the protocol, comparing it with respect to the expected behavior of finding more and more similar peers during its continuous execution. Significance weighting is assumed for all experiments involving the computation of recommendations.

6.2 Recommendation Quality

Next, we look at the *hit-rate* score, which help us evaluate whether the system is making recommendations for items that the peers will recognize and value.

The hit-rate for the pure-CF recommender implementation is presented in Figure 5.

In looking at the figure one can observe how the recommendation quality improves over time, as a consequence of the intra-neighborhood similarity improvement. The series shows that for the Swarmix architecture, the *hit-rate* measure is nearly equal to the central server's.

6.3 Failures and Voluntary-Leavings

We analyze the effect of peer failures and voluntary leavings from the overlay. For these experiments we applied a disturbance at cycle 50 and let the Swarmix network evolve. Finally, we report the *hit-rate* values corresponding to the

[2]Actually we dropped all ratings of movies that could not be identified in the Amazon taxonomy by Ziegler et al. [22], as we plan to compare with taxonomy-based recommendation strategies in upcoming work.

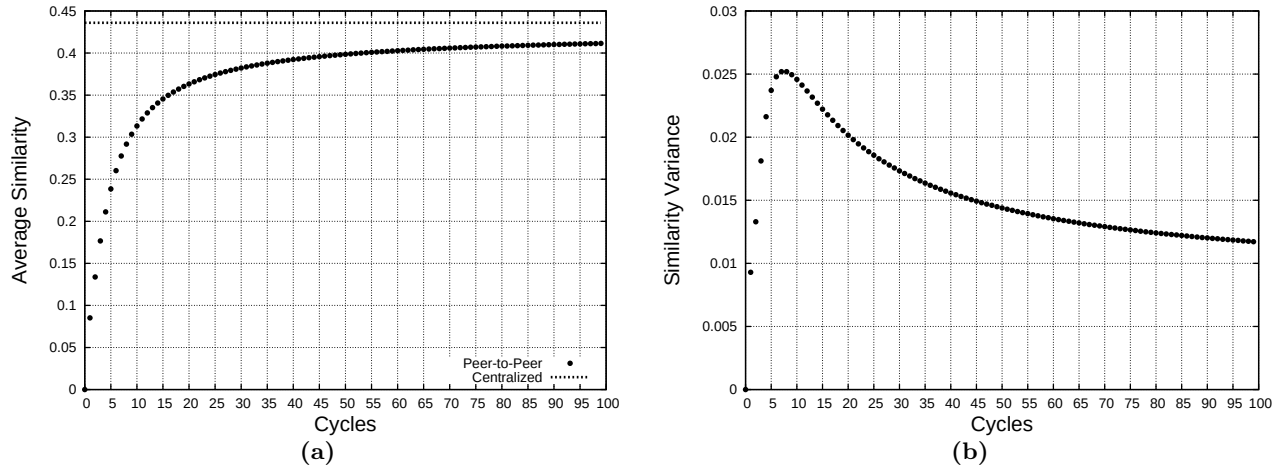

Figure 4: (a) Average similarity, (b) similarity variance.

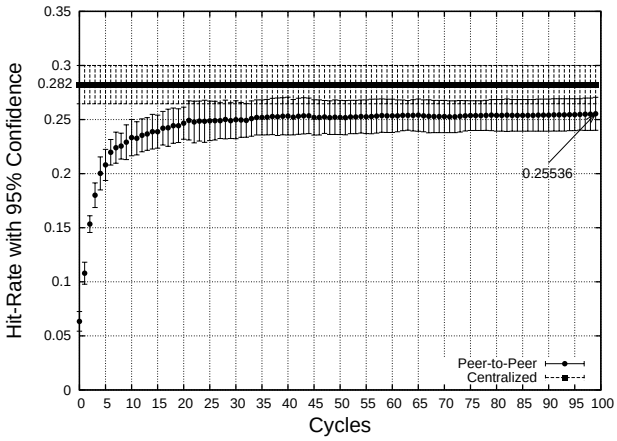

Figure 5: Hit-Rate

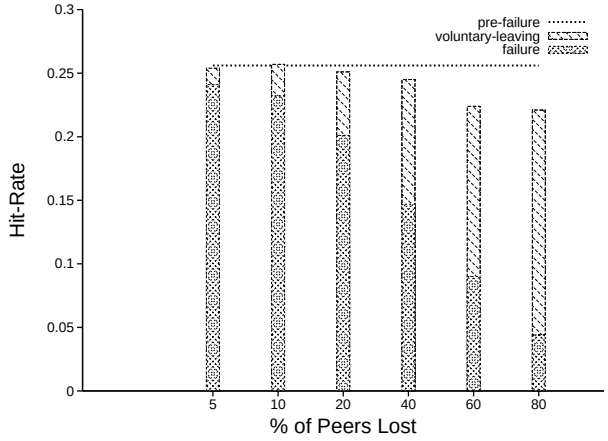

Figure 6: Failure and voluntary-leaving effect.

ones at cycle 100. Peer losses of 5%, 10%, 20%, 40%, 60% and 80% were simulated.

Failures. We perform these experiments considering that a peer v, disconnected from the network as a consequence of a failure, is not able to receive recommendations, but still wants to receive them. Therefore, we consider the total number of peers (i.e., 943) when computing the *hit-rate*.

Voluntary leavings. In case of a peer leaving the network voluntarily, we modified the *hit-rate* to take into consideration only those peers that remain connected to the overlay. If L represents the set of peers that have left the network, the hit-rate for voluntary leavings is computed as

$$\text{hit-rate}_{\text{voluntary-leavings}} := \frac{\sum_{v \in (V \setminus L)} hit(v)}{|V \setminus L|}$$

Therefore, we assumed that peers leaving the network do not want to receive their recommendations anymore. Note that this is a worst case scenario, because they are able to receive recommendations, locally computed from the cache entries, even in the case when no connection to the overlay exists (i.e., using their cache entries).

Figure 6 shows the simulation results.

6.4 Network Characteristics

Complex networks are frequently used to model a wide variety of systems of high technological, social, biological and intellectual importance. Ordinarily, the connection topology of such network models is assumed to be either completely regular or completely random, but many social, biological and technological network systems lie somewhere in between these two extremes. Watts and Strogatz [20] dubbed these network systems as *small-world* networks, in analogy with the small world phenomenon [15].

The small-world property appears to characterize most complex networks, such as electrical power grids, neural networks, science collaboration graphs and so forth [20, 1].

Two characteristics set small-world networks apart: first, a *small average path length*, typical of random graphs, defined as the number of edges in the shortest path between two vertices, averaged over all pairs of vertices; and it measures the typical separation between two vertices in the graph (a *global* property). Second, a *large clustering coefficient,*

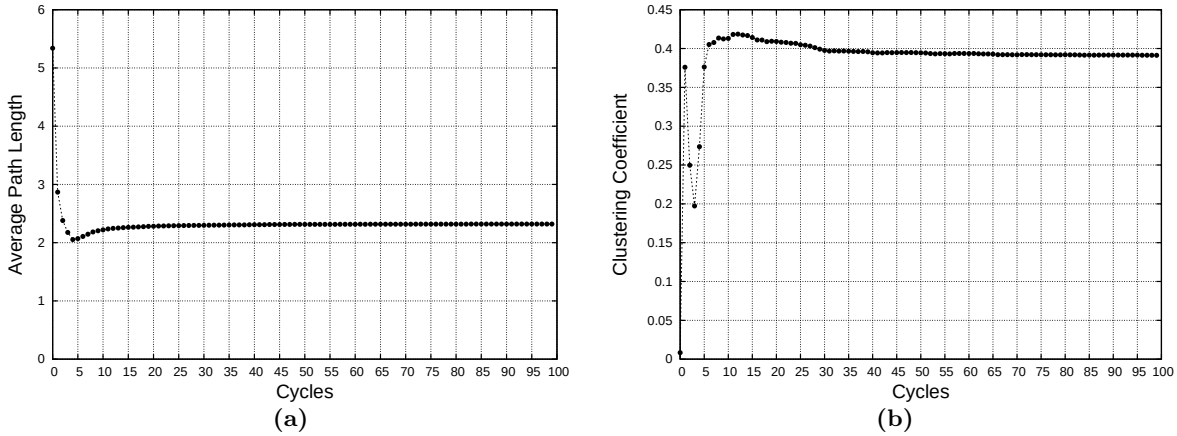

Figure 7: (a) Average path length, (b) clustering coefficient.

independent of network size, measures the cliquishness of a typical neighborhood (a *local* property), (i.e., how many of a node's neighbors are connected to each other).

We can observe from Figure 7(a) that very low average path lengths are obtained. Note that in order to get a finite value, the network has to be connected.

The *average clustering coefficient*, taken over all nodes in the overlay, is shown in Figure 7(b). Observe that the values shown are relatively high (if our networks were random, the clustering coefficient would be expected to be approximately $(k/n) = (20/943) = 0.02121$ [20, 1]).

The behavior during the initial cycles is explained by the application of the bootstrapping procedure previously explained. However, few cycles after starting to perform the Swarmix epidemic protocol, the average path length and clustering coefficient values converge quickly to their eventual values.

The small average path length values, in combination with the relatively high clustering coefficients, allow us to conclude that our Swarmix P2P overlay is a *small-world* network. This means that information dissemination is efficient, for two arbitrary nodes are separated by a few links only. Furthermore, it is expected that those high clustering coefficient values would provide certain resilience for network partitioning, e.g. in the presence of peer failures.

Note that these properties are not maintained explicitly, but they *emerge* from the underlying simple epidemic-style Swarmix protocol.

7. RELATED WORK

In this section, we present some examples of related research on deploying recommender systems in distributed architectures.

PocketLens [16] is a P2P-based collaborative filtering algorithm that incrementally updates an item-item model [7] for later use to make recommendations. In contrast to PocketLens, Swarmix builds a *user-based* matrix [12] for each peer v, where the users in the matrix correspond to v's neighbors only, avoiding scalability problems when the amount of users in the network increases.

Haase et al. [11] deploy a CF recommender system over a P2P-based personal bibliography management tool. The recommender system assists users in the management and evolution of their personal ontology by providing detailed suggestions of ontology changes. These suggestions are based on the usage information of the individual ontologies across the P2P network. Swarmix is domain-independent and could be tuned to deliver recommendations of actions, not only items, only requiring a meaningful way to represent user profiles in order to compute their similarity for neighborhood formation.

An entirely distributed CF algorithm called PipeCF, based on a content-addressable distributed hash table (DHT) infrastructure, is presented in [17]. Swarmix depends on a epidemic-style protocol for information dissemination.

One area of research that intersects with peer-to-peer recommender systems systems is that of mobile and intelligent software agents. Yenta [9], for example, is a decentralized multi-agent system that focuses on the issue of finding other peers with similar interests using referrals from other agents.

8. CONCLUSION AND FUTURE WORK

In this paper, we presented a distributed architecture for recommender systems, based upon epidemic-style protocols and geared towards peer-to-peer scenarios such as the Semantic Web, the Grid, and so forth. Our main focus in marrying these two worlds, i.e., the client/server-dominated world of recommender systems and the emerging peer-to-peer paradigm, into one coherent model has been to preserve the benefits of both, namely recommendation quality and scalability, robustness, and resilience to failure.

To this end, we empirically evaluated our architecture, assessing neighborhood similarity and the quality of recommendations first. Our results showed that our distributed architecture is able to maintain neighborhoods whose intra-similarity steadily improves over time. Final recommendations obtained are comparable to the ones expected in a centralized architecture, an implicit consequence of these aforementioned high-quality neighborhoods.

We also investigated characteristics of the evolving network, observing the emergence of a small-world network from its initial random-graph topology [8]. Consequently, we derive that information dissemination over the Swarmix network is efficient, for any two nodes are separated by small

geodesic distances only. Moreover, we expect those relatively high clustering coefficients $C(p)$ to provide substantial resilience against network partitioning, e.g., in case of peer failures. We also examined the network behavior in the presence of voluntary leavings, and failures.

Our research breaks new grounds and enters territory that still appears largely untouched. Hence, there remain numerous branches for future research to take.

First, owing to the very nature of the epidemic protocol combined with our *similarity*-based weighting function, our current setup risks the perpetual dissemination of outdated profile information. Assigning timestamps to profiles and incorporating an "aging" mechanism solves this issue in an efficient and simple manner. Furthermore, we also have to consider the dynamics of user preferences over time.

Second, we would like to better understand and characterize the social network characteristics of the emerging web. We already know about its small-world traits, i.e., high clustering coefficient, and small path lengths between any two peers [20]. We are now interested in its evolutionary *dynamics*, i.e., investigating the presence or absence of typical "rich get richer phenomena" [3], leading to power-law distributions of node degrees.

Other, more practical and user-centric considerations include privacy and security issues. Though being key motivations for crafting decentralized recommender systems, they have not been addressed in this paper. To this end, we believe that computational trust and trust propagation models [21] inherently solve the issue of privacy, and security in particular, and could likewise serve as efficient and scalable means for neighborhood selection.

We envision our architecture as part of the Semantic Web's infrastructure, with peers being represented by Semantic Web agents. Owing to different domains that users are interested in, e.g., not only books but also music, movies, etc., several instances of the protocol are supposed to run simultaneously in one single agent, allowing the computation of high-quality recommendations for each of these domains of interest.

9. REFERENCES

[1] R. Albert and A.-L. Barabasi. Statistical mechanics of complex networks. *CoRR*, cond-mat/0106096, 2001.

[2] R. A. Baeza-Yates and B. Ribeiro-Neto. *Modern Information Retrieval*. Addison-Wesley Longman Publishing Co., Inc., 1999.

[3] A.-L. Barabási and R. Albert. Emergence of scaling in random networks. *Science*, 286:509–512, October 1999.

[4] T. Berners-Lee, J. Hendler, and O. Lassila. The Semantic Web. *Scientific American*, May 2001.

[5] J. S. Breese, D. Heckerman, and C. M. Kadie. Empirical analysis of predictive algorithms for collaborative filtering. In *UAI*, pages 43–52, 1998.

[6] A. Demers, D. Greene, C. Houser, W. Irish, J. Larson, S. Shenker, H. Sturgis, D. Swinehart, and D. Terry. Epidemic Algorithms for Replicated Database Maintenance. *SIGOPS Oper. Syst. Rev.*, 22(1):8–32, 1988.

[7] M. Deshpande and G. Karypis. Item-based top-n recommendation algorithms. *ACM Trans. Inf. Syst.*, 22(1):143–177, 2004.

[8] P. Erdős and A. Rényi. On random graphs. *Publicationes Mathematicae*, 5:290–297, 1959.

[9] L. N. Foner. Yenta: a multi-agent, referral-based matchmaking system. In *AGENTS '97: Proceedings of the first international conference on Autonomous agents*, pages 301–307. ACM Press, 1997.

[10] GroupLens. MovieLens Dataset. Retrieved from http://www.grouplens.org/data/ on August 2004.

[11] P. Haase, A. Hotho, L. Schmidt-Thieme, and Y. Sure. Usage-driven Evolution of Personal Ontologies. *Proceedings of the 3rd International Conference on Universal Access in Human-Computer Interaction, Las Vegas, Nevada, USA*, July 2005.

[12] J. Herlocker and J. Konstan. An Empirical Analysis of Design Choices in Neighborhood-based Collaborative Filtering Algorithms. *Information Retrieval, 5 . Kluwer Academic Publishers*, pages 287–310, 2002.

[13] M. Jelasity, W. Kowalczyk, and M. van Steen. Newscast computing. Internal Report IR-CS-006, Vrije Universiteit Amsterdam, Department of Computer Science, Amsterdam, The Netherlands, Nov. 2003.

[14] M. Jelasity and M. van Steen. Large-scale newscast computing on the Internet. Internal Report IR-503, Vrije Universiteit Amsterdam, Department of Computer Science., Amsterdam, The Netherlands, Oct. 2002.

[15] S. Milgram. The small world problem. *Psychology Today*, pages 60 – 67, May 1967.

[16] B. N. Miller, J. A. Konstan, and J. Riedl. PocketLens: Toward a personal recommender system. *ACM Transactions on Information Systems (TOIS)*, 22(3):437–476, July 2004.

[17] Peng Han, Bo Xie, Fang Yang, and Ruimin Shen. A Scalable P2P recommender system based on distributed collaborative filtering. In *International Journal of Expert Systems with Applications*, volume 27, pages 203–210, Netherlands, Aug. 2004. Elsevier Publishers.

[18] B. M. Sarwar, G. Karypis, J. A. Konstan, and J. Riedl. Analysis of recommendation algorithms for e-commerce. In *ACM Conference on Electronic Commerce*, pages 158–167, 2000.

[19] Tomas Olsson. Bootstrapping and Decentralizing Recommender Systems. *Licentiate thesis 2003-006. Uppsala University and SICS.*, 2003.

[20] D. Watts and S. Strogatz. Collective Dynamics of "Small World" Networks. *Nature*, (393):440–442, 1998.

[21] C.-N. Ziegler and G. Lausen. Spreading activation models for trust propagation. In *Proceedings of the IEEE International Conference on e-Technology, e-Commerce, and e-Service*, Taipei, Taiwan, March 2004. IEEE Computer Society Press.

[22] C.-N. Ziegler, G. Lausen, and L. Schmidt-Thieme. Taxonomy-driven computation of product recommendations. In *CIKM '04: Proceedings of the Thirteenth ACM conference on Information and knowledge management*, pages 406–415. ACM Press, 2004.