Similitude: Decentralised Adaptation in Large-Scale P2P Recommenders

Davide Frey, Anne-Marie Kermarrec, Christopher Maddock, Andreas Mauthe, Pierre-Louis Roman, François Taïani

Similitude: Decentralised Adaptation in Large-Scale P2P Recommenders

Davide Frey[1], Anne-Marie Kermarrec[1], Christopher Maddock[2], Andreas Mauthe[2], Pierre-Louis Roman[3], and François Taïani[3]

[1] Inria, Rennes, France
[2] School of Computing and Communications, Lancaster University, UK
[3] Université de Rennes 1, IRISA – ESIR, France
{davide.frey, anne-marie.kermarrec}@inria.fr
{c.maddock1, a.mauthe}@lancaster.ac.uk
{pierre-louis.roman, francois.taiani}@irisa.fr

Abstract. Decentralised recommenders have been proposed to deliver privacy-preserving, personalised and highly scalable on-line recommendations. Current implementations tend, however, to rely on a hard-wired similarity metric that cannot adapt. This constitutes a strong limitation in the face of evolving needs. In this paper, we propose a framework to develop dynamically adaptive decentralised recommendation systems. Our proposal supports a *decentralised* form of adaptation, in which individual nodes can independently select, and update their own recommendation algorithm, while still collectively contributing to the overall system's mission.

Keywords: Distributed Computing, Decentralised Systems, Collaborative Filtering, Recommendation Systems, Adaptation

1 Introduction

With the growth of the modern web, recommendation has emerged as a key service to help users navigate today's on-line content. Designing highly scalable and privacy-preserving recommenders is hard, and one promising approach consists in exploiting fully decentralised mechanisms such as gossip [21, 5], or DHTs [29]. These decentralised recommenders, however, have so far used a mostly homogeneous design. They typically rely on one similarity metric [30] to self-organise large numbers of users in implicit communities and offer powerful means to compute personalised recommendations. Figuring out the right similarity metric that best fits the needs of a large collection of users is, however, highly challenging.

To address this challenge, we explore, in this paper, how dynamic adaptation can be applied to large-scale decentralised recommenders by allowing each individual node to choose autonomously between different similarity metrics. Extending on earlier works in the field [19, 12], we propose several adaptation variants, and show how small changes in adaptation decisions can drastically impact a recommender's overall performance, while demonstrating the feasibility of decentralised self-adaptation in peer-to-peer recommender systems.

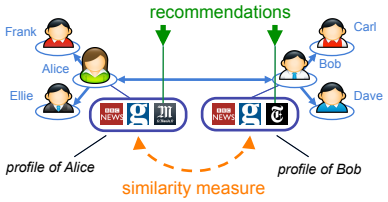

Fig. 1. Implicit overlay

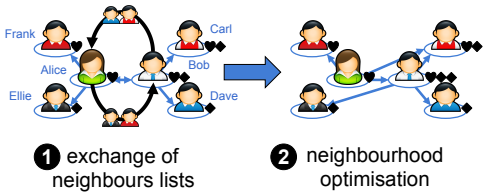

Fig. 2. Refining neighbourhoods

In the following, we motivate and present our work (Sec. 2 and 3), evaluate it (Sec. 4 and 5), before discussing related work (Sec. 6), and concluding (Sec. 7).

2 Background

Modern on-line recommenders [20, 11, 28, 22, 9] remain, in their vast majority, based around centralised designs. Centralisation comes, however, with two critical drawbacks. It first raises the spectre of a *big-brother* society, in which a few powerful players are able to analyse large swaths of personal data under little oversight. It also leads to a *data siloing* effect. A user's personal information becomes scattered across many competing services, which makes it very difficult for users themselves to exploit their data without intermediaries [31].

These crucial limitations have motivated research on decentralised recommendation systems [5, 7, 4, 25], in particular based on implicit interest-based *overlays* [30]. These overlays organise users (represented by their machines, also called *nodes*) into implicit communities to compute recommendations in a fully decentralised manner.

2.1 Interest-based Implicit Overlays

More precisely, these overlays seek to connect users[4] with their k most similar other users (where k is small) according to a predefined *similarity metric*. The resulting *k-nearest-neighbour* graph or $k$nn is used to deliver personalised recommendations in a scalable on-line manner. For instance, in Figure 1, *Alice* has been found to be most similar to *Frank*, *Ellie*, and *Bob*, based on their browsing histories; and *Bob* to *Carl*, *Dave*, and *Alice*.

Although *Bob* and *Alice* have been detected to be very similar, their browsing histories are not identical: *Bob* has not visited *Le Monde*, but has read the *New York Times*, which Alice has not. The system can use this information to recommend the *New York Times* to *Alice*, and reciprocally recommend *Le Monde* to Bob, thus providing a form of decentralised collaborative filtering [13].

Gossip algorithms based on asynchronous rounds [10, 30] turn out to be particularly useful in building such interest-based overlays. Users typically start

[4] In the following we will use *user* and *node* interchangeably.

with a random neighbourhood, provided by a random peer sampling service [17]. They then repeatedly exchange information with their neighbours, in order to improve their neighbourhood in terms of similarity. This greedy sampling procedure is usually complemented by considering a few random peers (returned by a decentralised *peer sampling service* [17]) to escape local minima.

For instance, in Figure 2, *Alice* is interested in hearts, and is currently connected to *Frank*, and to *Ellie*. After exchanging her neighbour list with *Bob*, she finds out about *Carl*, who appears to be a better neighbour than *Ellie*. As such, *Alice* replaces *Ellie* with *Carl* in her neighbourhood.

2.2 Self-Adaptive Implicit Overlays

The overall performance of a service using a $k$nn overlay critically depends on the similarity metric it uses. Unfortunately, deciding at design time which similarity metric will work best is highly challenging. The same metric might not work equally well for all users [19]. Further, user behaviour might evolve over time, thereby rendering a good initial static choice sub-efficient.

Instead of selecting a static metrics at design time, as most decentralised recommenders do [5, 3, 4], we propose to investigate whether each node can identify an optimal metric dynamically, *during* the recommendation process. Adapting a node's similarity metric is, however, difficult for at least three reasons. First, nodes only possess a limited view of the whole system (their *neighbourhood*) to make adaptation decisions. Second, there is a circular dependency between the information available to nodes for adaptation decisions and the actual decision taken. A node must rely on its neighbourhood to decide whether to switch to a new metric. But this neighbourhood depends on the actual metric being used by the node, adding further instability to the adaptation. Finally, because of the decentralised nature of these systems, nodes should adapt independently of each other, in order to limit synchronisation and maximise scalability.

3 Decentralised Adaptation

We assume a peer-to-peer system in which each node p possesses a set of *items*, items(p), and maintains a set of k neighbours ($k = 10$ in our evaluation). p's neighbours are noted $\Gamma(p)$, and by extension, $\Gamma^2(p)$ are p's neighbours' neighbours. Each node p is associated with a similarity metric, noted p.sim, which takes two sets of items and returns a similarity value.

The main loop of our algorithm (dubbed SIMILITUDE) is shown in Alg. 1 (when executed by node p). Ignoring line 3 for the moment, lines 2-4 implement the greedy $k$nn mechanism presented in Section 2. At line 4, $\mathbf{argtop}^k$ selects the k nodes of *cand* (the candidate nodes that may become p's new neighbours) that maximise the similarity expression $p.\text{sim}\big(\text{items}(p), \text{items}(q)\big)$.

Recommendations are generated at lines 5-6 from the set it_Γ of items of all users in p's neighbourhood (noted $\text{items}\big(\Gamma(p)\big)$). Recommendations are ranked using the function SCORE at line 8, with the similarity score of the user(s) they

Algorithm 1 SIMILITUDE

1: **in every round do**
2: $cand \leftarrow \Gamma(p) \cup \Gamma^2(p) \cup 1$ random node
3: ADAPTSIM($cand$)
4: $\Gamma(p) \leftarrow$ $\mathbf{argtop}^k_{q \in cand} \left(p.\text{sim}(\text{items}(p), \text{items}(q)) \right)$
5: $it_\Gamma \leftarrow \text{items}(\Gamma(p)) \setminus \text{items}(p)$
6: $rec \leftarrow$ $\mathbf{argtop}^m_{i \in it_\Gamma} \left(\text{SCORE}(i, p.\text{sim}, \text{items}(p), \Gamma(p)) \right)$
7: **end round**
8: **function** SCORE($i, sim, items, \Gamma$)
9: **return** $\sum_{q \in \Gamma \mid i \in \text{items}(q)} sim(items, \text{items}(q))$
10: **end function**

Algorithm 2 ADAPTSIM

1: **function** ADAPTSIM($cand$)
2: $top\_sims \leftarrow$ $\mathbf{argmax}_{s \in SIM} \left(\mathbf{avg}_4 \left(\text{EVAL\_SIM}(s, cand) \right) \right)$
3: **if** $p.\text{sim} \notin top\_sims$ **then**
4: $p.\text{sim} \leftarrow$ random element from top_sims
5: **end if**
6: **end function**
7: **function** EVAL_SIM($s, cand$)
8: $hidden_f \leftarrow$ proportion f of items(p)
9: $visible_f \leftarrow \text{items}(p) \setminus hidden_f$
10: $\Gamma_f \leftarrow \mathbf{argtop}^k_{q \in cand} \left(s(visible_f, \text{items}(q)) \right)$
11: $it_f \leftarrow \text{items}(\Gamma_f) \setminus visible_f$
12: $rec_f \leftarrow$ $\mathbf{argtop}^m_{i \in it_f} \left(\text{SCORE}(i, s, visible_f, \Gamma_f) \right)$
13: **return** $S = \frac{|rec_f \cap hidden_f|}{|rec_f|}$
14: **end function**

are sourced from. Recommendations suggested by multiple users take the sum of all relevant scores. The top m recommendations from it_Γ (line 6) are suggested to the user (or all of them if there are less than m).

3.1 Dynamic Adaptation of Similarity

The adaptation mechanism we propose (ADAPTSIM) is called at line 3 of Alg. 1, and is shown in Alg. 2. A node p estimates the potential of each available metric ($s \in SIM$, line 2) using the function EVAL_SIM(s). In EVAL_SIM(s), p hides a fraction f of its own items (lines 8-9) and creates a 'temporary potential neighbourhood' Γ_f for each similarity metric available (line 10, $f = 20\%$ in our evaluation). From each temporary neighbourhood, p generates a set of recommendations (lines 11-12) and evaluates them against the fraction f of internally hidden items, resulting in a score S for each similarity s (its *precision* (Figure 5)).

This evaluation is repeated four times and averaged to yield a set of the highest-achieving metrics (top_sims) (note that multiple metrics may achieve the same score). If the current metric-in-use $p.\text{sim}$ is not in top_sims, p switches to a random metric from top_sims (lines 3-4).

After selecting a new metric, a node suspends the metric-selection process for two rounds during which it only refines its neighbours. This *cool-off* period allows the newly selected metric to start building a stable neighbourhood thereby limiting oscillation and instability.

3.2 Enhancements to Adaptation Process

We now extend the basic adaptation mechanism presented in Section 3.1 with three additional *modifiers* that seek to improve the benefit estimation, and limit instability and bias: *detCurrAlgo*, *incPrevRounds* and *incSimNodes*.

detCurrAlgo (short for "detriment current algorithm") slightly detracts from the score of the current metric in use. This modifier tries to compensate for the fact that metrics will always perform better in neighbourhoods they have built up themselves. In our implementation, the score of the current metric in use is reduced by 10%.

incPrevRounds (short for "incorporate previous rounds") takes into consideration the scores S_{r-i} obtained by a metric in previous rounds to compute a metric's actual score in round r, $S^{\star}_{r}$ (Figure 3). In doing so, it aims at reducing the bias towards the current environment, thereby creating a more stable network with respect to metric switching.

incSimNodes (short for "incorporate similar nodes") prompts a node to refer to the metric choice of the most similar nodes it is aware of in the system. This is based on the knowledge that similar metrics are preferable for nodes with similar profiles, and thus if one node has discovered a metric which it finds to produce highly effective results, this could be of significant interest to other similar nodes. The modifier works by building up an additional score for each metric, based on the number of nodes using the same metric in the neighbourhood. This additional score is then balanced with the average of the different metrics' score (Figure 4).

4 Evaluation Approach

We validate our adaptation strategies by simulation. In this section, we describe our evaluation protocol; we then present our results in Section 5.

4.1 Data Sets

We evaluate SIMILITUDE on two datasets: Twitter, and MovieLens. The former contains the feed subscriptions of 5,000 similarly-geolocated Twitter users, randomly selected from the larger dataset presented in [8][5]. Each user has a profile containing each of her Twitter subscriptions, i.e., each subscribed feed counts as a positive rating. The MovieLens dataset [1] contains 1 million movie ratings from 6038 users, each consisting of an integer value from 1 to 5. We count values 3 and above as positive ratings. We pre-process each dataset by first removing the items with less than 20 positive ratings because they are of little interest to the recommendation process. Then, we discard the users with less than five remaining ratings. After pre-processing, the Twitter dataset contains 4569 users with a mean of 105 ratings per user, while the MovieLens dataset contains 6017 users with a mean of 68 ratings per user.

[5] An anonymised version of this dataset is available at `http://ftaiani.ouvaton.org/ressources/onlyBayLocsAnonymised_21_Oct_2011.tgz`

$$S_r^\star = S_r + \sum_{i=1}^{5}(0.5 - \tfrac{i}{10}) \times S_{r-i}^\star$$

Fig. 3. Incorporating previous rounds

$$S_{r,sim}^\star = S_{r,sim} + \underset{x \in SIM}{avg}(S_{r,x}) \times \tfrac{|sim\ in\ \Gamma(p)|}{|\Gamma(p)|}$$

Fig. 4. Incorporating similar nodes

$$Precision(u_i \in users) = \tfrac{|rec_i \cap hidden_i|}{|rec_i|}$$

$$Recall(u_i \in users) = \tfrac{|rec_i \cap hidden_i|}{|hidden_i|}$$

Fig. 5. Precision and recall

$$Overlap(u_i, u_j) = |items_i \cap items_j|$$

$$Big(u_i, u_j) = |items_j|$$

$$OverBig(u_i, u_j) = Overlap(u_i, u_j) + Big(u_i, u_j)$$

$$Jaccard(u_i, u_j) = \tfrac{Overlap(u_i, u_j)}{items_i \cup items_j}$$

Fig. 6. The four similarity metrics used

4.2 Evaluation Metrics

We evaluate recommendation quality using precision and recall (Figure 5). *Precision* measures the ability to return few incorrect recommendations, while *recall* measures the ability to return many correct recommendations. In addition, we evaluate specific aspects of our protocol. First, we count how many nodes reach their *optimal* similarity metrics—we define more precisely what we understand by *optimal* in Section 4.5. Finally, we observe the level of instability within the system, by recording the number of nodes that switch metric during each round.

4.3 Simulator and Cross Validation

We measure recommendation quality using a cross-validation approach. We split the profile of each user into a visible item set containing 80% of its items, and a hidden item set containing the remaining 20%. We use the visible item set to construct the similarity-based overlay and as a data source to generate recommendations as described in Section 3. We then consider a recommendation as successful if the hidden item set contains a corresponding item.

In terms of protocol parameters, we randomly associate each node with an initial neighbourhood of 10 nodes, as well as with a randomly selected similarity metric to start the refinement process. At each round, the protocol provides each node with a number of suggestions equal to the average number of items per user. We use these suggestions to compute precision and recall. Each simulation runs for 100 rounds; we repeat each run 10 times and average the results. Finally, we use two rounds of cool-off by default.

4.4 Similarity Metrics

We consider four similarity metrics: *Overlap*, *Big*, *OverBig* and, *Jaccard* [19], shown in Figure 6. These metrics are sufficiently different to represent distinct similarity choices for each node, and offer a representative adaptation scenario.

Overlap counts the items shared by a user and its neighbour. As such, it tends to favour users with a large number of items. *Big* simply counts the number of items of the neighbour, presuming that the greater the number of items available,

the more likely a match is to be found somewhere in the list. This likewise favours users with a larger number of items. *OverBig* works by combining *Big* and *Overlap*—thereby discrediting the least similar high-item users. Finally *Jaccard* normalises the overlap of items by dividing it by the total number of items of the two users; it therefore provides improved results for users with fewer items.

It is important to note that the actual set of metrics is not our main focus. Rather, we are interested in the adaptation process, and seek to improve recommendations by adjusting the similarity metrics of individual nodes.

4.5 Static Metric Allocations

We compare our approach to *six* static (i.e., non-adaptive) system configurations, which serve as baselines for our evaluation. In the first four, we statically allocate the same metric to all nodes from the set of metrics in Figure 6 (*Overlap*, *Big*, *OverBig*, and *Jaccard*). These baselines are static and homogeneous.

The fifth (*HeterRand*) and sixth (*HeterOpt*) baselines attempt to capture two extreme cases of heterogeneous allocation. *HeterRand* randomly associates each node with one of the four above metrics. This configuration corresponds to a system that has no a-priori knowledge regarding optimal metrics, and that does not use dynamic adaptation. *HeterOpt* associates each node with its *optimal* similarity metric. To identify this optimal metric, we first run the first four baseline configurations (static and homogeneous metrics). For each node, *HeterOpt* selects one of the metrics for which the node obtains the highest average precision. *HeterOpt* thus corresponds to a situation in which each node is able to perfectly guess which similarity metric works best for itself.

5 Experimental Results

5.1 Static Baseline

We first determine the set of optimal metrics for each node in both datasets as described in Section 4.5. To estimate variability, we repeat each experiment twice, and compare the two sets of results node by node. 43.75% of the nodes report the same optimal metrics across both runs. Of those that do not, 35.43% list optimal metrics that overlap across the two runs. In total, 79.18% of nodes' optimal metrics match either perfectly or partially across runs. Figure 7 depicts the distribution obtained in the first run for both datasets.

5.2 Basic Similitude

We first test the basic SIMILITUDE with no modifiers, and a cool-off period of two rounds. Figures 8 and 9 present precision and recall (marked SIMILITUDE (BASIC)). Figure 10 depicts the number of users selecting one of the optimal metrics, while Figure 11 shows the switching activity of users.

These results show that SIMILITUDE allows nodes to both find their optimal metric and switch to it. Compared to a static random allocation of metrics

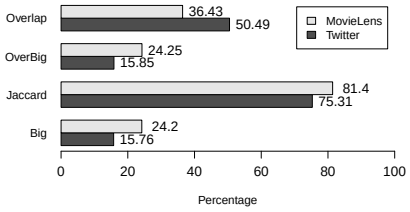

Fig. 7. Distribution of optimal metrics

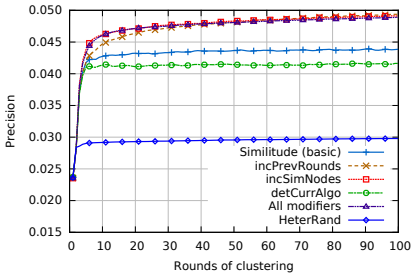

Fig. 8. Precision

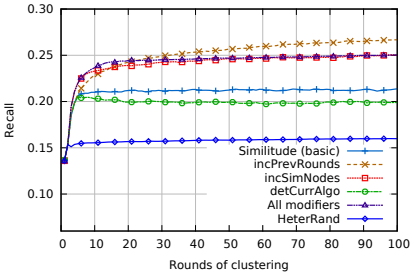

Fig. 9. Recall

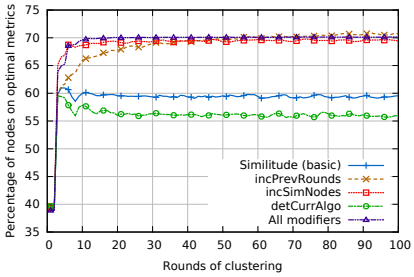

Fig. 10. Nodes optimal metrics

(*HeterRand*), SIMILITUDE improves precision by 47.22%, and recall by 33.75%. A majority of nodes (59.55%) reach their optimal metrics, but 17.43% remain unstable and keep switching metrics throughout the experiment.

5.3 Effects of the Modifiers

detCurrAlgo has a negative effect on every aspect: precision, recall, number of nodes on optimal metrics, and stability (Figures 8 through 11). Precision and recall decrease by 5.08% and 6.93% respectively compared to basic SIMILITUDE. At the same time, the final number of nodes on their optimal metrics decreases by 6.02%, and unstable nodes increase by 35.29%.

This shows that, although reducing the current metric's score might intuitively make sense because active metrics tend to shape a node's neighbourhood to their advantage, this modifier ends up disrupting the whole adaptation process. We believe this result depends on the distribution of optimal metrics (Figure 7). Since one metric is optimal for a majority of nodes, reducing its score only causes less optimal metrics to take over. It would be interesting to see how this modifier behaves on a dataset with a more balanced distribution of optimal metrics than the two we consider here.

incPrevRounds, unlike *detCurrAlgo*, increases precision by 12.57% and recall by 24.75% with respect to basic SIMILITUDE, while improving stability by 55.92%. The number of nodes reaching an optimal metric improves by 18.81%.

As expected, *incPrevRounds* greatly improves the stability of the system; it even enhances every evaluation metric we use. The large reduction in the number of unstable nodes, and the small increase in that of nodes reaching their optimal metrics suggest that *incPrevRounds* causes nodes to settle on a metric faster, whether or not that metric is optimal. One possible explanation is that, if one metric performs especially well in a round with a particular set of neighbours, all future rounds will be affected by the score of this round.

incSimNodes, like *incPrevRounds*, improves basic SIMILITUDE on every aspect. Precision increases by 11.91%, recall by 16.88%, the number of nodes on their optimal metrics by 16.53%, and that of unstable nodes decreases by 49.26%.

With this modifier, most of the nodes switch to the same similarity metric (*Jaccard*). Since *incSimNodes* tends to boost the most used metric in each node's neighbourhood, it ends up boosting the most used metric in the system, creating a snowball effect. Given that *Jaccard* is the optimal metric for most of the nodes, it is the one that benefits the most from *incSimNodes*.

Even if completely different by design, both *incPrevRounds* and *incSimNodes* have very similar results when tested with Twitter. This observation cannot be generalised as the results are not the same with MovieLens (Figures 15 and 16).

All modifiers activates all three modifiers with the hope of combining their effects. Results show that this improves precision and recall by 29.11% and 43.99% respectively. The number of nodes on optimal metrics also increases by 32.51%. Moreover none of the nodes switch metrics after the first 25 rounds.

Activating all the modifiers causes most nodes to employ the metric that is optimal for most nodes in Figure 7, in this case *Jaccard*. This explains why no node switches metrics and why the number of nodes reaching optimal metrics (70.15%) is very close to the number of nodes with *Jaccard* as an optimal metric (75.31%). The difference gets even thinner without cool-off (Section 5.5): 73.43% of the nodes use their optimal metrics.

5.4 Weighting the Modifiers

We balance the effect of the two additive modifiers (*incPrevRounds* and *incSimNodes*) by associating each of them with a multiplicative weight. A value of 0 yields the basic SIMILITUDE, a value of 1 applies the full effect of the modifier, while a value of 0.5 halves its effect. We use a default weight of 0.5 for both of them because they perform best with this value when operating together.

Figure 12 shows the precision and recall of *incPrevRounds* and *incSimNodes* with their respective weights ranging from 0 (basic SIMILITUDE) to 1, with a 0.1 step. *incPrevRounds* peaks at a weight of 0.5 even when operating alone, while *incSimNodes* peaks at 0.7, but it still performs very well at 0.5.

5.5 Varying the Cool-Off Period

As described in Section 3.1, the cool-off mechanism seeks to prevent nodes from settling too easily on a particular metric. To assess the sensitivity of this param-

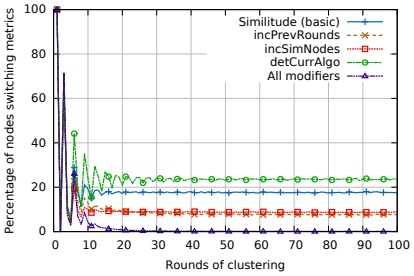

Fig. 11. Switching activity

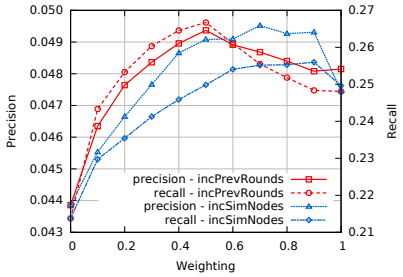

Fig. 12. Weighting *incPrevRounds* and *incSimNodes*

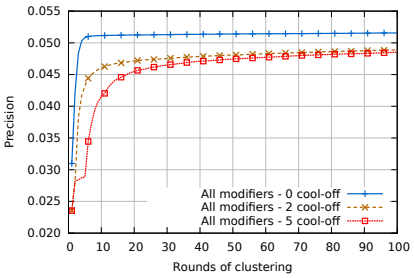

Fig. 13. Precision under different cool-off

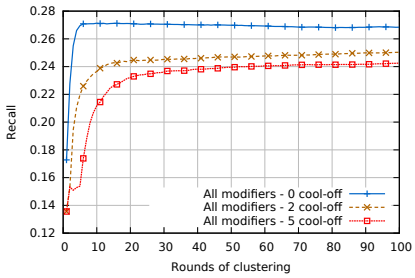

Fig. 14. Recall under different cool-off

eter, Figures 13 and 14 compare the results of SIMILITUDE with all the modifiers, when the cool-off period varies from 0 (no cool-off) to 5 rounds.

Disabling cool-off results in a slight increase in precision (5.45%) and in recall (7.23%) when compared to 2 rounds of cool-off. Optimal metrics are reached by 3.73% more nodes, and much faster, attaining up to 73.43% nodes. Removing cool-off reduces a metric's ability to optimise a neighbourhood to its advantage, as there is only a single round of clustering before the metric is tested again. While cool-off can offer additional stability in adaptive systems, the stability provided by the modifiers appears to be sufficient in our model. Cool-off, instead, leads metrics to over-settle, and produces a negative effect.

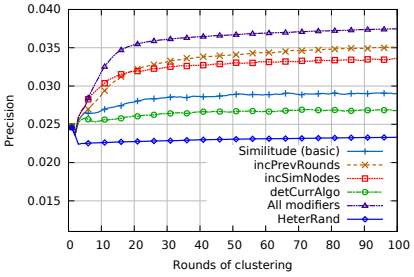

Fig. 15. Precision (MovieLens)

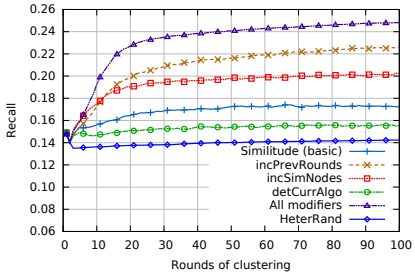

Fig. 16. Recall (MovieLens)

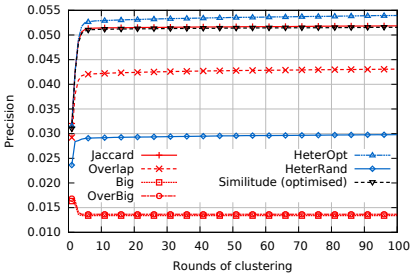

Fig. 17. SIMILITUDE against static solutions (Twitter)

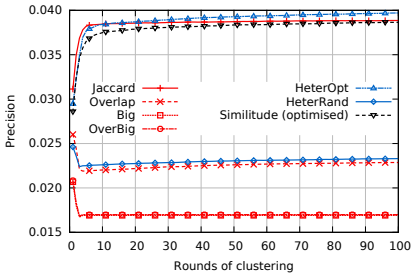

Fig. 18. SIMILITUDE against static solutions (MovieLens)

5.6 MovieLens Results

Figures 15 and 16 show the effect of SIMILITUDE on precision and recall with the different modifiers using the MovieLens dataset. The results are similar to those obtained with Twitter (Figures 8 and 9).

As with the Twitter dataset, basic SIMILITUDE outperforms *HeterRand* in precision by 24.52% and in recall by 21.02%. By the end of the simulations, 59.95% of the nodes reach an optimal metric and 15.73% still switch metrics.

The behaviour of the modifiers compared to basic SIMILITUDE is also similar. *detCurrAlgo* degrades precision by 7.58%, recall by 9.86%, the number of nodes on optimal metrics by 7.07%, and the number of nodes switching metrics by 33.40%. *incPrevRounds* improves precision by 21.02%, recall by 31.19%, the number of nodes on optimal metrics by 20.52%, and the number of nodes switching metrics by 62.08%. *incSimNodes* improves precision by 15.75%, recall by 17.38%, the number of nodes on optimal metrics by 15.12%, and the number of nodes switching metrics by 28.61%.

All modifiers improves precision by 29.11%, recall by 43.99%, the number of nodes on optimal metrics by 32.51%, and there are no nodes switching metrics after the first 25 rounds. As with the Twitter dataset, activating all the modifiers makes all the nodes use the similarity metric which is optimal for the majority of the system: *Jaccard*. The number of nodes reaching optimal metrics (79.44%) and the number of nodes with *Jaccard* as optimal metric (81.40%) are almost identical. Without cool-off, SIMILITUDE even reaches 80.57% nodes on an optimal metric, getting even closer to that last number.

5.7 Complete System

We now compare the results of the best SIMILITUDE variant (all the modifiers and 0 cool-off, noted SIMILITUDE (OPTIMISED)) with the six static configurations we introduced in Section 4.5.

For the Twitter dataset, Figure 17 shows that our adaptive system outperforms the static random allocation of metrics by 73.06% in precision, and overcomes all but one static homogeneous metrics, *Jaccard*, which is on par with

SIMILITUDE (OPTIMISED). For the MovieLens dataset, Figure 18 shows very similar results where SIMILITUDE (OPTIMISED) has a higher precision than *HeterRand* by 65.85%, is on par with *Jaccard*, and has a slightly lower precision than *HeterOpt* (−2.6%). Selecting *Jaccard* statically would however require knowing that this metric performs best, which may not be possible in evolving systems (in which *Jaccard* might be replaced by another metric as users' behaviours change).

5.8 Discussion

SIMILITUDE (OPTIMISED) enables a vast majority of the nodes (73.43% for Twitter, 80.57% for MovieLens) to eventually switch to an optimal metric, which corresponds to the number of nodes having *Jaccard* as their optimal metric (Figure 7). By looking at these number, we can say that our system has the ability to discover which metric is the best suited for the system without needing prior evaluation. While this already constitutes a very good result, there remains a difference between SIMILITUDE and *HeterOpt* (the optimal allocation of metrics to nodes), which represents the upper bound that a dynamically adaptive system might be able to reach. Although achieving the performance of a perfect system might prove unrealistic, we are currently exploring potential improvements.

First, *incSimNodes* could be reworked in order to have a more balanced behaviour to avoid making the whole system use only one similarity metric, even if it is the most suited one for the majority of the nodes. Next, we observe that nodes appear to optimise their neighbourhood depending on their current metric, as opposed to basing their metric choice on their neighbourhood. This may lead to local optima because metrics perform notably better in neighbourhoods they have themselves refined. Our initial attempt at avoiding such local optima with the *detCurrAlgo* proved unsuccessful, but further investigation could result in rewarding future work. For example, we are considering decoupling the choice of the metric from the choice of the neighbourhood. Nodes may compare the performance of metrics using randomly selected neighbourhoods, and then move to the clustering process only using the best-performing metric.

Finally, it would be interesting to see how *detCurrAlgo*, *incSimNodes* and more generally SIMILITUDE behave on a dataset with a more balanced distribution of optimal metrics since their effects and results highly depend on it.

6 Related Work

Several efforts have recently concentrated on decentralised recommenders [14, 24, 2, 6, 27] to investigate their advantages in terms of scalability and privacy. Earlier approaches exploit DHTs in the context of recommendation. For example, PipeCF [14] and PocketLens [24] propose Chord-based CF systems to decentralise the recommendation process on a P2P infrastructure. Yet, more recent solutions have focused on using randomised and gossip-based protocols [5, 18, 4].

Recognised as a fundamental tool for information dissemination [16, 23], Gossip protocols exhibit innate scalability and resilience to failures. As they copy

information over many links, gossip protocols generally exhibit high failure resilience. Yet, their probabilistic nature also makes them particularly suited to applications involving uncertain data, like recommendation.

Olsson's *Yenta* [26] was one of the first systems to employ gossip protocols in the context of recommendation. This theoretical work enhances decentralised recommendation by taking trust between users into account. *Gossple* [5] uses a similar theory to enhance navigation through query expansion and was later extended to news recommendation [7]. Finally, in [15], Hegedűs et al. present a gossip-based learning algorithm that carries out 'random walks' through a network to monitor concept drift and adapt to change in P2P data-mining.

7 Conclusion

We have presented SIMILITUDE, a decentralised overlay-based recommender that is able to adapt at runtime the similarity used by individual nodes. SIMILITUDE demonstrates the viability of decentralised adaptation for very large distributed systems, and shows it can compete against static schemes.

Although promising, our results shows there is still room for improvement. In particular, we would like to see how a dataset with a more balanced distribution of optimal metrics affects SIMILITUDE and its modifiers. We also think that the *detCurrAlgo* and *incSimNodes* modifiers could benefit from further improvements, and thus bring the performance of SIMILITUDE closer to that of a static optimal-metric allocation.

Acknowledgements

This work was partially funded by the French National Research Agency (ANR) project *SocioPlug* - ANR-13-INFR-0003 (`http://socioplug.univ-nantes.fr`).

References

1. Movielens 1 million ratings dataset. `http://grouplens.org/datasets/movielens`.
2. Tribler. `http://www.tribler.org`.
3. X. Bai, M. Bertier, R. Guerraoui, A.-M. Kermarrec, and V. Leroy. Gossiping personalized queries. In *EDBT'2010*.
4. R. Baraglia, P. Dazzi, M. Mordacchini, and L. Ricci. A peer-to-peer recommender system for self-emerging user communities based on gossip overlays. *J. of Comp. and Sys. Sciences*, 2013.
5. M. Bertier, D. Frey, R. Guerraoui, A.-M. Kermarrec, and V. Leroy. The gossple anonymous social network. In *Middleware'2010*.
6. A. Boutet, D. Frey, R. Guerraoui, A. Jégou, and A.-M. Kermarrec. Privacy-Preserving Distributed Collaborative Filtering. In *NETYS'2014*.
7. A. Boutet, D. Frey, R. Guerraoui, A. Jégou, and A.-M. Kermarrec. WhatsUp Decentralized Instant News Recommender. In *IPDPS*, 2013.
8. J. Carretero, F. Isaila, A.-M. Kermarrec, F. Taïani, and J. M. Tirado. Geology: Modular georecommendation in gossip-based social networks. In *ICDCS 2012*.

9. A. S. Das, M. Datar, A. Garg, and S. Rajaram. Google news personalization: scalable online collaborative filtering. In *WWW*, 2007.
10. A. Demers, D. Greene, C. Hauser, W. Irish, J. Larson, S. Shenker, H. Sturgis, D. Swinehart, and D. Terry. Epidemic Algorithms for Replicated Database Maintenance. In *PODC'87*.
11. Facebook Inc. Facebook: Company info – statistics. `https://newsroom.fb.com/company-info/`, March 2014. Accessed: 2014-05-13.
12. D. Frey, A.-M. Kermarrec, C. Maddock, A. Mauthe, and F. Taïani. Adaptation for the masses: Towards decentralized adaptation in large-scale p2p recommenders. In *13th Workshop on Adaptive & Reflective Middleware*, ARM '14.
13. D. Goldberg, D. Nichols, B. M. Oki, and D. Terry. Using collaborative filtering to weave an information tapestry. *CACM*, 1992.
14. P. Han, B. Xie, F. Yang, and R. Shen. A scalable p2p recommender system based on distributed collaborative filtering. *Expert Systems with Applications*, 2004.
15. I. Hegedus, R. Ormándi, and M. Jelasity. Gossip-based learning under drifting concepts in fully distributed networks. In *SASO 2012*.
16. M. Jelasity, A. Montresor, and O. Babaoglu. T-man: Gossip-based fast overlay topology construction. *Computer networks*, 53(13):2321–2339, 2009.
17. M. Jelasity, S. Voulgaris, R. Guerraoui, A.-M. Kermarrec, and M. van Steen. Gossip-based peer sampling. *ACM TOCS*, 25, 2007.
18. A.-M. Kermarrec, V. Leroy, A. Moin, and C. Thraves. Application of random walks to decentralized recommender systems. In *OPODIS'10*.
19. A.-M. Kermarrec and F. Taïani. Diverging towards the common good: heterogeneous self-organisation in decentralised recommenders. In *SNS'2012*.
20. J. A. Konstan, B. N. Miller, D. Maltz, J. L. Herlocker, L. R. Gordon, and J. Riedl. Grouplens: Applying collaborative filtering to usenet news. *CACM*.
21. V. Leroy, B. B. Cambazoglu, and F. Bonchi. Cold start link prediction. In *KDD'2010*.
22. G. Linden, B. Smith, and J. York. Amazon. com recommendations: Item-to-item collaborative filtering. *IEEE Internet Computing*, 2003.
23. G. Mega, A. Montresor, and G. P. Picco. Efficient dissemination in decentralized social networks. In *IEEE P2P 2011*.
24. B. N. Miller, J. A. Konstan, and J. Riedl. Pocketlens: toward a personal recommender system. *TOIS*, 2004.
25. A. Moreno, H. Castro, and M. Riveill. Decentralized recommender systems for mobile advertisement. In *Workshop on Personalization in Mobile Applications (PEMA'11)*, Chicago, Illinois, USA, Oct. 2011. ACM.
26. T. Olsson. Decentralised social filtering based on trust. In *AAAI-98 Recommender Systems Workshop*.
27. V. Schiavoni, E. Rivière, and P. Felber. Whisper: Middleware for confidential communication in large-scale networks. In *ICDCS 2011*, June 2011.
28. Y. Song, S. Dixon, and M. Pearce. A survey of music recommendation systems and future perspectives. In *CMMR'2012*.
29. J. M. Tirado, D. Higuero, F. Isaila, J. Carretero, and A. Iamnitchi. Affinity p2p: A self-organizing content-based locality-aware collaborative peer-to-peer network. *Comp. Net.*, 54, 2010.
30. S. Voulgaris and M. v. Steen. Epidemic-style management of semantic overlays for content-based searching. In *Euro-Par'05*.
31. C.-m. A. Yeung, I. Liccardi, K. Lu, O. Seneviratne, and T. Berners-Lee. Decentralization: The future of online social networking. In *W3C Workshop on the Future of Social Networking*, 2009.