D.1.3 – Protocols for emergent localities



ANR-13-INFR-0003
socioplug.univ-nantes.fr

Davide Frey[1,2], Achour Mostéfaoui[3], Matthieu Perrin[3], Anne-Marie Kermarrec[2], Christopher Maddock[4], Andreas Mauthe[4], Pierre-Louis Roman[1,2,5], François Taïani[1,2,5]

[1]*IRISA Rennes*
[2]*Inria Rennes - Bretagne Atlantique*
[3]*LINA, Université de Nantes*
[4]*School of Computing and Communications, Lancaster University, UK*
[5]*Université de Rennes 1 - ESIR*
Email contact: {davide.frey,michel.raynal,francois.taiani}@irisa.fr, matthieu.perrin@etu.univ-nantes.fr, achour.mostefaoui@univ-nantes.fr

---

This report presents two contributions that illustrate the potential of emerging-locality protocols in large-scale decentralized systems, in two areas of decentralized social computing: recommendation, and eventual consistency of mutable data structures.

The first contribution consists of a framework supporting the development of dynamically adaptive decentralised recommendation systems. 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. Our framework address this through 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.

Our second contribution addresses the growing demand for differentiated consistency requirements in large-scale applications. A large number of today's applications rely on Eventual Consistency, a consistency model that emphasizes liveness over safety. Designers generally adopt this consistency model uniformly throughout a distributed system due to its ability to scale as the number of users or devices grows larger. But this clashes with the need for differentiated consistency requirements. In this contribution, we address this need by introducing *UPS*, a novel consistency mechanism that offers differentiated eventual consistency and delivery speed by working in pair with a two-phase epidemic broadcast protocol. We propose a closed-form analysis of our approach's delivery speed, and we evaluate our complete protocol experimentally on a simulated network of one million nodes. To measure the consistency trade-off, we formally define a novel and scalable consistency metric operating at runtime.

Keywords: Decentralized distributed systems, recommenders, eventual consistency

---

1 Introduction

Modern distributed computer systems have reached sizes and extensions not envisaged even a decade ago: modern datacenters routinely comprise tens of thousands of machines [VPK+15], and on-line applications are typically distributed over several of these datacenters into complex geo-distributed infrastructures [gda, LVA+15]. Such enormous scales come with a host of challenges that distributed-system researchers have focused on over the last four decades.

The intuition underlying the SOCIOPLUG project is that decentralization can help address some of these challenges by offering inherent scalability properties to key services of modern distributed systems. Organizing a decentralized system is however neither an easy, nor a straightforward task: constructing a global knowledge regarding the entire system's current state is extremely costly, and in some cases not even possible. This difficulty lies in the intrinsic cost of communication in a large-scale distributed system. Messages take time to travel, might not arrive, or might arrive after unpredictable delays (asynchrony). Worse, if system nodes are subject to failures, it is usually not possible to distinguish between a crashed node and one hampered by very slow communications.

Decentralized systems address this challenge by exploiting local behaviors: they build on interactions that only involve a few nodes, which know each other usually through overlay views of limited size. Creating such *local neighborhoods* remains however often as much a craft as a science. In this report we discuss two recent contributions of the SOCIOPLUG project that aim to provide a more systematic way to construct such *localities* through emerging behaviors [FKM+15, FMP+16]:

- SIMILITUDE [FKM+15] considers the problem of decentralized recommendation in peer-to-peer systems, a critical component of our envisioned decentralized social computing ecosystem. Such decentralized recommenders usually rely on a similarity metric to implement a collaborative filtering strategy. Selecting which best similarity metric to use is however a difficult and complex task. In Section 2, we present a decentralized adaptive mechanism by which each node may dynamically select the similarity metric that seems to best support its recommending goals, thereby giving rise to a form of *emerging similarity field*.
- In a second contribution, *UPS* [FMP+16], we look at the problem of eventual consistency in large scale systems. Ensuring that all nodes perceive the same consistent state of a common shared data is often unpractical in such systems, a limitation that has prompted researchers to propose an array of weak consistency conditions that allow the local states of replicated data structures to diverge under well-defined rules. One highly popular such constraint is *eventual consistency*, which ensures that during periods of stability the whole system progressively converges back to a global coherent state. Current eventual consistency protocols unfortunately offer nowadays "one-size-fits-all" guarantees, and do not distinguish between nodes with distinct consistency requirements within the same system. In Section 3 we present a novel consistency mechanism, termed *UPS* (for *Update-Query Consistency with Primaries and Secondaries*), that provides different levels of eventual consistency within the same system. *UPS* combines the update-query consistency protocol proposed in [PMJ15] with a two-phase epidemic broadcast protocol (called *GPS*) involving two types of nodes: *Primary* and *Secondary*. *Primary* nodes (the *elite*) seek to receive object modifications as fast as possible while *Secondary* nodes (the *masses*) strive to minimize the amount of transient inconsistencies they perceive (Figure 20).

Both contributions illustrate the capabilities that a generic service for emerging localities could provide, paving the ways towards such a mechanism for large-scale decentralized systems.

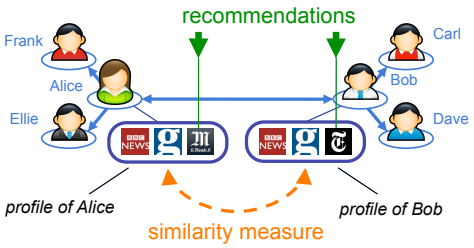


Figure 1: Implicit overlay

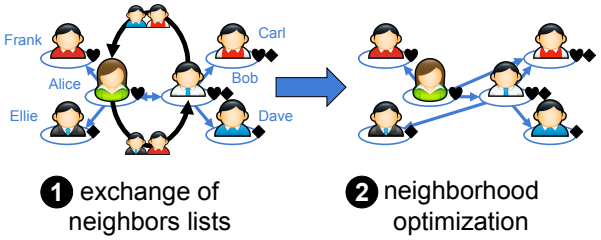


Figure 2: Refining neighbourhoods

2 Similitude: Decentralised Adaptation in Large-Scale P2P Recommenders

2.1 Background

Modern on-line recommenders [KMM+97, Fac14, SDP, LSY03, DDGR07] 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 [YLL+09].

These crucial limitations have motivated research on decentralised recommendation systems [BFG+a, BFG+13, BDMR13, MCR11], in particular based on implicit interest-based *overlays* [VS]. These overlays organise users (represented by their machines, also called *nodes*) into implicit communities to compute recommendations in a fully decentralised manner.

2.1.1 Interest-based Implicit Overlays

More precisely, these overlays seek to connect users† 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 [GNOT92].

Gossip algorithms based on asynchronous rounds [DGH+, VS] turn out to be particularly useful in building such interest-based overlays. Users typically start with a random neighbourhood, provided by a random peer sampling service [JVG+07a]. 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* [JVG+07a]) 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.

† In the following we will use *user* and *node* interchangeably.

2.1.2 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 [KT]. 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 [BFG+a, BBG+, BDMR13], 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.

2.2 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 2.1. At line 4, **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 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).

2.2.1 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 6-7) and creates a 'temporary potential neighbourhood' Γ_f for each similarity metric available (line 8, $f = 20\%$ in our evaluation). From each temporary neighbourhood, p generates a set of recommendations (lines 9-10) 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.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.

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: $\Gamma(p) \leftarrow$ **argtop**$^k_{q \in cand}\left(p.\text{sim}\big(\text{items}(p), \text{items}(q)\big)\right)$
5: $it_\Gamma \leftarrow \text{items}\big(\Gamma(p)\big) \setminus \text{items}(p)$
6: $rec \leftarrow$ **argtop**$^m_{i \in it_\Gamma}\left(\text{SCORE}\big(i, p.\text{sim}, \text{items}(p), \Gamma(p)\big)\right)$
7: **end round**
8: **function** SCORE($i, sim, items, \Gamma$)
9: **return** $\sum_{q \in \Gamma | i \in \text{items}(q)} sim(items, \text{items}(q))$

Algorithm 2 ADAPTSIM

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

2.2.2 Enhancements to Adaptation Process

We now extend the basic adaptation mechanism presented in Section 2.2.1 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).

2.3 Evaluation Approach

We validate our adaptation strategies by simulation. In this section, we describe our evaluation protocol; we then present our results in Section 2.4.

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

Figure 3: Incorporating previous rounds

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

Figure 4: Incorporating similar nodes

2.3.1 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 [CIK+]‡. Each user has a profile containing each of her Twitter subscriptions, i.e., each subscribed feed counts as a positive rating. The MovieLens dataset [mov] 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.

2.3.2 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 2.3.5. Finally, we observe the level of instability within the system, by recording the number of nodes that switch metric during each round.

2.3.3 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 2.2. 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.

2.3.4 Similarity Metrics

We consider four similarity metrics: *Overlap*, *Big*, *OverBig* and, *FrereJacc* [KT], 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 in the list.

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

$$\begin{aligned} Precision(u_i \in users) &= \frac{|rec_i \cap hidden_i|}{|rec_i|} \\ Recall(u_i \in users) &= \frac{|rec_i \cap hidden_i|}{|hidden_i|} \end{aligned}$$

Figure 5: Precision and recall

$$\begin{aligned} Overlap(u_i, u_j) &= |items_i \cap items_j| \\ Big(u_i, u_j) &= |items_j| \\ OverBig(u_i, u_j) &= Overlap(u_i, u_j) + Big(u_i, u_j) \\ FrereJacc(u_i, u_j) &= \frac{Overlap(u_i, u_j)}{|items_i| + |items_j|} \end{aligned}$$

Figure 6: The four similarity metrics used

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 *FrereJacc* 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. *FrereJacc*[§] consists of a variant of the well-known *Jaccard* similarity metric.

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.

2.3.5 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 *FrereJacc*). 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.

2.4 Experimental Results

2.4.1 Static Baseline

We first determine the set of optimal metrics for each node in both datasets as described in Section 2.3.5. 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.

2.4.2 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.

[§] *FrereJacc* was erroneously labeled as *Jaccard* in the proceedings version.

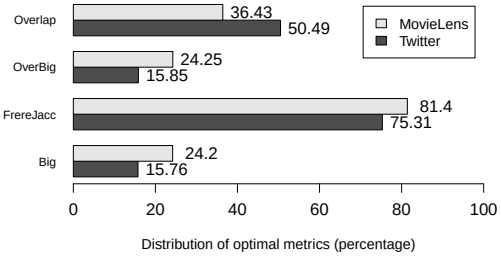


Figure 7: Distribution of optimal metrics

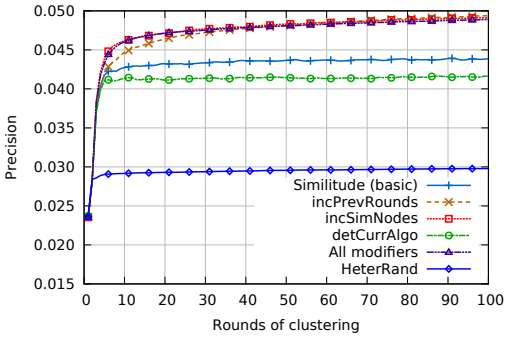


Figure 8: Precision

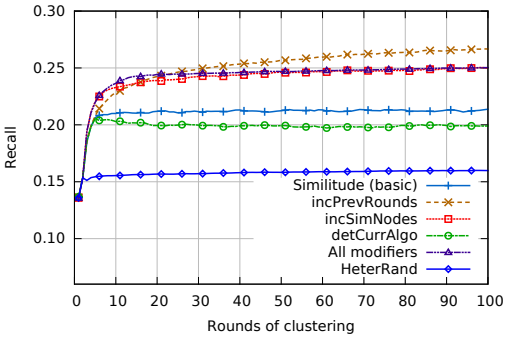


Figure 9: Recall

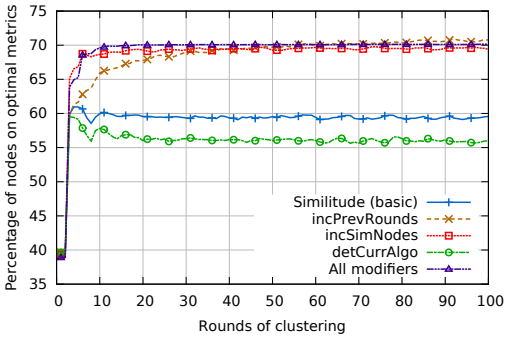


Figure 10: Nodes optimal metrics

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 (*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.

2.4.3 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 (*FrereJacc*). 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 *FrereJacc* 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 *FrereJacc*. 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 *FrereJacc* as an optimal metric (75.31%). The difference gets even thinner without cool-off (Section 2.4.5): 73.43% of the nodes use their optimal metrics.

2.4.4 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.

2.4.5 Varying the Cool-Off Period

As described in Section 2.2.1, the cool-off mechanism seeks to prevent nodes from settling too easily on a particular metric. To assess the sensitivity of this parameter, 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

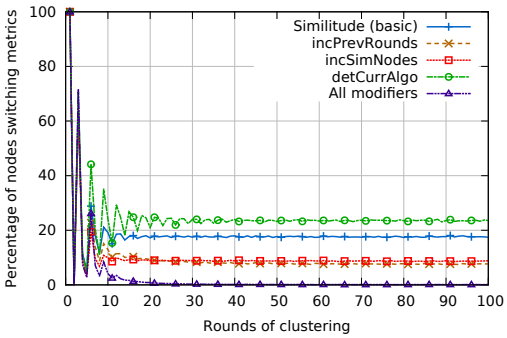


Figure 11: Switching activity

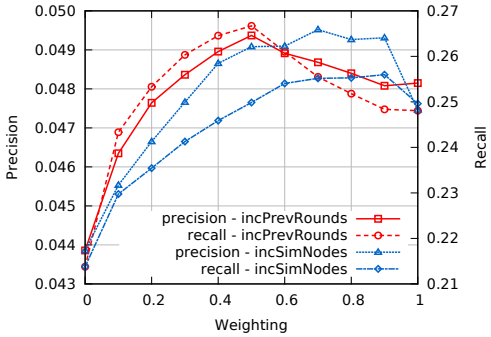


Figure 12: Weighting *incPrevRounds* and *incSimNodes*

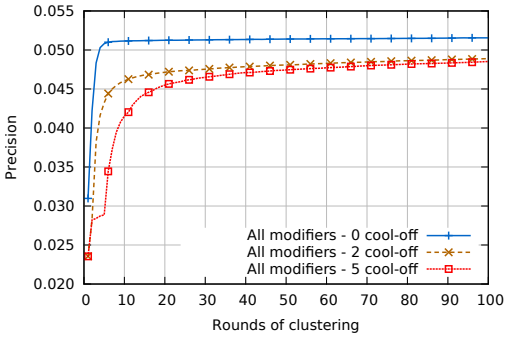


Figure 13: Precision under different cool-off

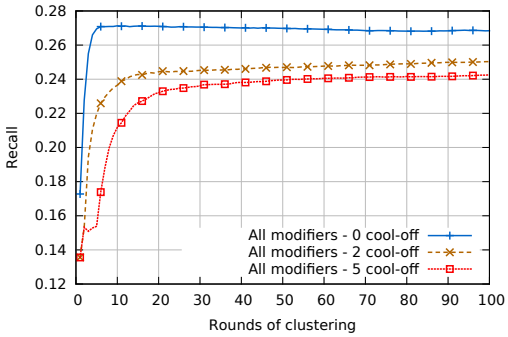


Figure 14: Recall under different cool-off

to over-settle, and produces a negative effect.

2.4.6 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: *FrereJacc*. The number of nodes reaching optimal metrics (79.44%) and the number of nodes with *FrereJacc* as optimal metric (81.40%) are almost identical.

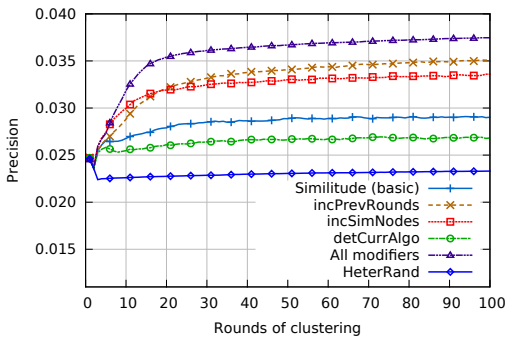


Figure 15: Precision (MovieLens)

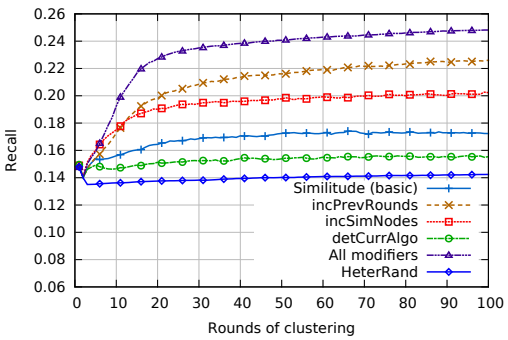


Figure 16: Recall (MovieLens)

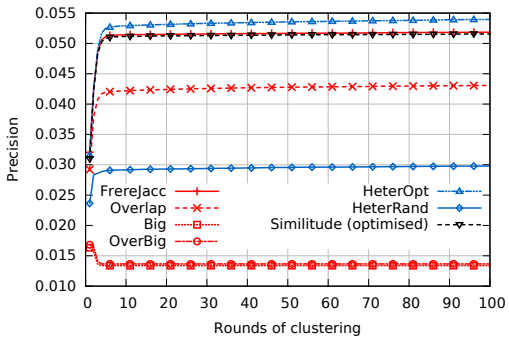


Figure 17: SIMILITUDE against static solutions (Twitter)

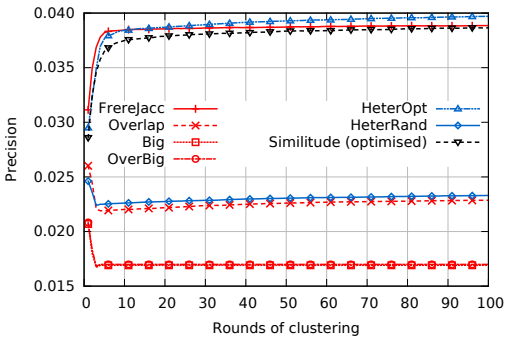


Figure 18: SIMILITUDE against static solutions (MovieLens)

Without cool-off, SIMILITUDE even reaches 80.57% nodes on an optimal metric, getting even closer to that last number.

2.4.7 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 2.3.5.

For the Twitter dataset, Figure 17 shows that our adaptive system out-performs the static random allocation of metrics by 73.06% in precision, and overcomes all but one static homogeneous metrics, *FrereJacc*, 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 *FrereJacc*, and has a slightly lower precision than *HeterOpt* (−2.6%). Selecting *FrereJacc* statically would however require knowing that this metric performs best, which may not be possible in evolving systems (in which *FrereJacc* might be replaced by another metric as users' behaviours change).

2.4.8 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 *FrereJacc* 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.

2.5 Related Work

Several efforts have recently concentrated on decentralised recommenders [HXYS04, MKR04, tri, BFG+b, SRF11] to investigate their advantages in terms of scalability and privacy. Earlier approaches exploit DHTs in the context of recommendation. For example, PipeCF [HXYS04] and PocketLens [MKR04] 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 [BFG+a, KLMT, BDMR13].

Recognised as a fundamental tool for information dissemination [JMB09, MMP], 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* [Ols] 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* [BFG+a] uses a similar theory to enhance navigation through query expansion and was later extended to news recommendation [BFG+13]. Finally, in [HOJ], 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.

3 *UPS*: differentiating eventual consistency in large-scale distributed systems

3.1 Motivation

One of the key challenges of building distributed systems arises from the inherent tension between fault-tolerance, performance, and consistency, elegantly captured by the CAP impossibility theorem [GL02]. As systems grow in size, the data they hold must be replicated for reasons of both performance (to mitigate the inherent latency of widely distributed systems) and fault-tolerance (to avoid service interruption in the presence of faults). Replicated data is unfortunately difficult to keep consistent: *strong consistency* (such as linearizability or sequential consistency) is particularly expensive to implement in large-scale systems, and cannot be simultaneously guaranteed

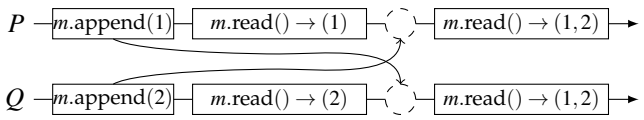


Figure 19: An eventually consistent append-only message queue.

together with availability, when using a realistic unreliable network [GL02].

The cost and limitations of strong consistency have prompted an increased interest in weaker consistency conditions for large scale systems, such as PRAM or causal consistency [BGHS13, ALR13, LFKA11]. Among these conditions, *eventual consistency* [Vog09, SPBZ11] aims to strike a balance between speed, dynamicity, and agreement within a system. Intuitively, eventual consistency allows the replicas of a distributed shared object to temporarily diverge, as long as they eventually converge back to a unique global state.

Formally, this global consistent state should be reached once updates on the object stop (with additional constraints usually linking the object's final value to its sequential specification) [PMJ15]. In a practical system, a consistent state should be reached every time updates stop for *long enough* [LVA+15]. How long is long enough depends on the properties of the underlying communication service, notably on its *latency* and *ordering guarantees*. These two key properties stand in a natural trade-off, in which latency can be traded off for better (probabilistic) ordering properties [MMF+15, BGL+06, SPMO02]. This inherent tension builds a picture in which an eventually consistent object must strike a compromise between speed (how fast are changes visible to other nodes) and consistency (to which extend do different nodes agree on the system's state).

Figure 19, for instance, shows the case of a distributed append-only message queue m manipulated by two processes P and Q. m supports two operations append(x), which appends an integer x to the queue, and read$()$, which returns the current content of m. In Figure 19, both P and Q eventually converge to the same consistent global state $(1,2)$, that includes both modifications m.append(1) by P and m.append(2) by Q, in this order. Q, however, experiences an intermediate inconsistent state when it reads (2): this read does not "see" the append operation by P which has been ordered before m.append(2), and is inconsistent with the final state $(1,2)$[¶]. Q could increase the odds of avoiding this particular inconsistency by delaying its first read operation, thus augmenting its chances of receiving information regarding P's append operation on time (dashed circle). Such delays improve consistency, but reduce the speed of change propagation across replicas, and must be chosen with care.

Most existing solutions to eventual consistency resolve this tension between speed and consistency by applying one trade-off point uniformly to all the nodes in a system [LVA+15, MMF+15]. However, as systems continue to grow in size and expand in geographic span, they become more diverse, and must cater for diverging requirements. In this report, we argue that this heterogeneity increasingly call for differentiated consistency levels in large scale systems. This observation has been made by other researchers, who have proposed a range of hybrid consistency conditions over the years [XSK+14, FRT15, Fri95, LPC+12], but none of them has so far considered how eventual consistency on its own could offer differentiated levels of speed and consistency within the same system.

Designing such a protocol raises however an important methodological point: how to measure consistency. Consistency conditions are typically formally defined as predicates on execution histories; a system execution is thus either consistent, or it is not. However, practitioners using eventual consistency are often interested in the current "level" of consistency of a live system, i.e. how far the system currently is from a consistent situation. Quantitatively measuring a system's incon-

[¶] This inconsistency causes in particular Q to observe an illegal state transition from (2) to $(1,2)$.

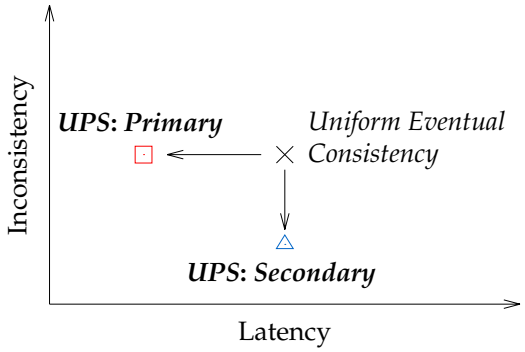


Figure 20: Aimed trade-off between consistency and speed in *UPS*

sistencies is unfortunately not straightforward: some practical works [LVA+15] measure the level of "agreement" between nodes, i.e. how many nodes see the same state, but this approach has little theoretical grounding and can thus lead to paradoxes. For instance, returning to Figure 19, if we assume a large number of nodes (e.g. $Q_1, Q_2, .., Q_n$) reading the same inconsistent state (2) as Q, the system will appear close to agreement (many nodes see the same state), although it is in fact largely inconsistent.

To address the above challenges, this report makes the following contributions:

- We propose a novel consistency mechanism, termed *UPS* (for *Update-Query Consistency with Primaries and Secondaries*), that provides different levels of eventual consistency within the same system (Sections 3.3.2,3.3.3). *UPS* combines the update-query consistency protocol proposed in [PMJ15] with a two-phase epidemic broadcast protocol (called *GPS*) involving two types of nodes: *Primary* and *Secondary*. *Primary* nodes (the *elite*) seek to receive object modifications as fast as possible while *Secondary* nodes (the *masses*) strive to minimize the amount of transient inconsistencies they perceive (Figure 20).
- We formally analyze the latency behavior of the *GPS*-part of *UPS* by providing closed-form approximations for the latency incurred by *Primary* and *Secondary* nodes. (Section 3.3.4)
- We introduce a novel consistency metric that allows us to quantify the amount of inconsistency experienced by *Primary* and *Secondary* nodes executing *UPS* (Section 3.4).
- We experimentally evaluate the performance of *UPS* by measuring its consistency and latency properties in large-scale simulated networks of 1M nodes (Section 3.5). We show in particular that the cost paid by each class of nodes is in fact very small compared to an undifferentiated system: *Primary* nodes experience similar levels of inconsistency as undifferentiated nodes with lower latency, while *Secondary* nodes observe less inconsistency at a minimal latency costs.

3.2 Background and Problem Statement

3.2.1 Update Consistency

As we hinted at, in Section 3.1, eventual consistency requires replicated objects to converge to a globally consistent state when update operations stop for "long enough". By itself, this condition turns out to be too weak as the convergence state does not need to depend on the operations

carried out on the object. For this reason, actual implementations of eventually consistent objects refine eventual consistency by linking the convergence state to its sequential specification.

In this report, we focus on one such refinement, *Update consistency* [PMJ15]. Let us consider the append-only queue object of Figure 19. Its sequential specification consists of two operations.

- $append(x)$, with $x \in \mathbb{Z}$, appends the value x at the end of the queue.
- $read()$, returns the sequence of all the elements ever appended, in their append order.

When multiple distributed agents update the queue, update consistency requires the final convergence state to be the result of a total ordering of all the append operations which respects the program order. For example, the scenario in Figure 19 satisfies update consistency because the final convergence state results from the ordering $m.append(1), m.append(2)$, which itself respects the program order. An equivalent definition [PMJ15] states that an execution history respects update consistency if it contains an infinite number of updates or if it is possible to remove a finite number of queries from it, so that the resulting pruned history is sequentially consistent. In Figure 19, removing $R(2)$ achieves this.

Algorithm 3 shows an algorithm from [PMJ15] that implements the update-consistent append-only queue of Figure 19. Unlike CRDTs [SPBZ11] that rely on commutative ("conflict-free") operations, Algorithm 3 exploits a broadcast operation together with Lamport Clocks, a form of logical time-stamps that makes it possible to reconstruct a total order of operations after the fact [PMJ15]. Relying on this after-the-fact total order allows *update consistency* to support non-commutative operations, like the queue in this case.

3.2.2 Problem Statement

The key feature of update consistency lies in the ability to define precisely the nature of the convergence state reached once all updates have been issued. However, the nature of intermediate states also has an important impact in practical systems. This raises two important challenges. First, existing systems address the consistency of intermediate states by implementing uniform constraints that all the nodes in a system must follow [BGYZ14]. But different actors in a distributed application may have different requirements regarding the consistency of these intermediate states. Second, even measuring the level of inconsistency of these states remains an open question. Existing systems-oriented metrics do not take into account the ordering of update operations (append in our case) [LVA+15, GLS11, GRA+14, PPR+11], while theoretical ones require global knowledge of the system [ZK12] which makes them impractical at large scale.

In this following sections, we address both of these challenges. First we propose a novel broadcast mechanism that, together with Algorithm 3, satisfies update consistency, while supporting differentiated levels of consistency for query operations that occur before the convergence state. Specifically, we exploit the evident trade-off between speed of delivery and consistency, and we target heterogeneous populations consisting of an elite of *Primary* nodes that should receive fast, albeit possibly inconsistent, information, and a mass of *Secondary* nodes that should only receive stable consistent information, albeit more slowly. Second, we propose a novel metric to measure the level of inconsistency of an append-only queue, and use it to evaluate our protocol.

3.3 The GPS broadcast protocol

3.3.1 System model

We consider a large set of nodes $p_1, ..., p_N$ that communicate using point-to-point messages. Any node can communicate with any other node, given its identifier. We use probabilistic algorithms

Algorithm 3 Update consistency for an append-only queue

```
1: variables
2:   int id                                              ▷ Node identifier
3:   set <int, int, V> U ← ∅                             ▷ Set of updates to the queue
4:   int clock_id ← 0                                    ▷ Node's logical clock
5: procedure APPEND(v)                                   ▷ Append a value v to the queue
6:   clock_id ← clock_id + 1
7:   U ← U ∪ {< clock_id, id, v >}
8:   BROADCAST (< clock_id, id, v >)
9: upon receive (< ck_msg, id_msg, v_msg >) do
10:  clock_id ←MAX(clock_id, ck_msg)
11:  U ← U ∪ {< ck_msg, id_msg, v_msg >}
12:
13: procedure READ( )                                   ▷ Read the current state of the queue
14:  q ← ()                                              ▷ Empty queue
15:  for all < clock_id, id, v > ∈ U sorted by (clock_id, id) do
16:    q ← q · v
17:  return q
```

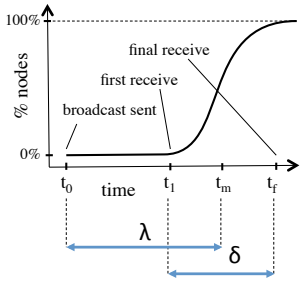


Figure 21: Two sorts of speeds: latency (λ) and jitter (δ)

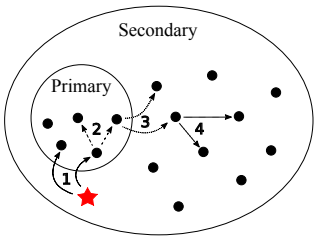


Figure 22: Model of *GPS* and path of an update in the system.

in the following that are naturally robust to crashes and message losses, but do not consider these aspects in the rest of the report for simplicity. Nodes are categorized in two classes: a small number of *Primary* nodes (the *elite*) and a large number of *Secondary* nodes (the *masses*). The class of a node is an application-dependent parameter that captures the node's requirements in terms of update query consistency: *Primary* nodes should perceive object modification as fast as possible, while *Secondary* nodes should experience as few inconsistencies as possible.

3.3.2 Intuition and overview

We have repeatedly referred to the inherent trade-off between speed and consistency in eventually consistent systems. On deeper examination, this trade-off might appear counter-intuitive: if *Primary* nodes receive updates faster, why should not they also experience higher levels of consistency? This apparent paradox arises because we have so far silently confused *speed* and *latency*. The situation within a large-scale broadcast is in fact more subtle and involves two sorts of speeds (Figure 21): *latency* (λ, shown as an average over all nodes in the figure) is the time a message m takes to reach individual nodes, from the point in time of m's sending (t_0). *Jitter* (δ), by contrast, is

the delay between the first (t_1) and the last receipt (t_f) of a broadcast. (In most large-scale broadcast scenarios, $t_0 - t_1$ is small, and the two notions tend to overlap.) Inconsistencies typically arise in Algorithm 3 when some updates have only partially propagated within a system, and are thus predominantly governed by the jitter δ rather than the average *latency* λ. The gossip-based broadcast protocol we propose, *Gossip Primary-Secondary* (*GPS*), exploits this distinction and reduces λ for *Primary* nodes (thus increasing the speed at which updates are visible), while reducing δ for *Secondary* nodes (thus increasing consistency, but at the cost of a slightly higher λ).

More precisely, *GPS* uses the set *Primary* nodes as a sort of message "concentrator" that accumulates copies of an update u before collectively forwarding it to *Secondary* nodes. The main phases of this sequence is shown in Figure 22:

1. A new update u is first sent to *Primary* nodes (**1**);
2. *Primary* nodes disseminate u among themselves (**2**);
3. Once most *Primary* nodes have received u, they forward it to *Secondary* nodes (**3**);
4. Finally, *Secondary* nodes disseminate u among themselves (**4**).

A key difficulty in this sequence consists in deciding when to switch from Phase **2** to **3**. A collective, coordinated transition would require some global synchronization mechanism, a costly and generally impracticable solution in a very large system. Instead, *GPS* relies on less accurate but more scalable local procedure based on broadcast counts, which allows each *Primary* to decide locally when to start forwarding to secondaries.

3.3.3 *The* GPS *algorithm*

The pseudo-code of *GPS* is shown in Algorithm 4. *GPS* follows the standard models of reactive epidemic broadcast protocols [KMG03, TLB]. Each node keeps a history of the messages received so far (in the R variable, line 8), and decide whether to re-transmit a received broadcast to fanout other nodes based on this history. Contrary to a standard epidemic broadcast, however, *GPS* handles *Primary* and *Secondary* nodes differently.

- *First*, *GPS* uses two distinct *Random Peer Sampling* protocols (RPS) [JVG+07b] (lines 10-11) to track the two classes of nodes. Both *Primary* and *Secondary* nodes use the RPS view of their category to re-transmit a message they receive for the first time to fanout other nodes in their own category (lines 23 and 24), thus implementing Phases **2** and **4**.
- *Second*, *GPS* handles retransmissions differently depending on a node's class (*Primary* or *Secondary*). *Primary* nodes use the inherent presence of message duplicates in gossip protocols, to decide locally when to switch from Phase **2** to **3**. More specifically, each node keeps count of the received copies of individual messages (lines 8, 13, 20). *Primary* nodes use this count to detect duplicates, and forward this message to fanout *Secondary* nodes (line 24) when a duplicate is received for the first time, thus triggering Phase **3**.

We can summarize the behavior of both classes by saying that *Primary* nodes *infect twice and die*, whereas *Secondary* nodes *infect and die*. For comparison, an standard *infect and die* gossip without classes (called *Uniform Gossip* in the following), is shown in Algorithm 5. We will use *Uniform Gossip* as our baseline for our analysis (Section 3.3.4) and our experimental evaluation (Section 3.5).

Algorithm 4 – *Gossip Primary-Secondary* for a node

```
1: parameters
2:   integer fanout                          ▷ Number of nodes to send to
3:   integer rpsViewSize                     ▷ Out-degree per class of each node
4:   boolean isPrimary                       ▷ Class of the node
5: initialization
6:   set{node} Γ_P ← ∅                       ▷ Set of Primary neighbors
7:   set{node} Γ_S ← ∅                       ▷ Set of Secondary neighbors
8:   map{message, int} R ← ∅                 ▷ Counters of message
                                             ▷ duplicates received
9: periodically
10:  Γ_P ← rpsViewSize nodes from Primary-RPS
11:  Γ_S ← rpsViewSize nodes from Secondary-RPS
12: procedure BROADCAST(msg)                 ▷ Called by the application
13:  R ← R ∪ {(msg, 1)}
14:  GOSSIP(msg, Γ_P)
15: procedure GOSSIP(msg, targets)
16:  for all j ∈ {fanout random nodes in targets} do
17:   SENDTONETWORK(msg, j)
18: upon receive (msg) do
19:  counter ← R[msg] + 1
20:  R ← R ∪ {(msg, counter)}
21:  if counter = 1 then DELIVER(msg)        ▷ Deliver 1st receipt
22:  if isPrimary then
23:   if counter = 1 then GOSSIP(msg, Γ_P)
24:   if counter = 2 then GOSSIP(msg, Γ_S)
25:  else
26:   if counter = 1 then GOSSIP(msg, Γ_S)
```

3.3.4 Analysis of GPS

In the following we compare analytically the expected performance of *GPS* and compare it to *Uniform Gossip* in terms of message complexity and latency.

Our analysis uses the following parameters:

- Network N of size: $|N| \in \mathbb{N}$.
- Fanout: $f \in \mathbb{N}$.
- Density of *Primary* nodes: $d \in \mathbb{R}_{[0,1]}$ (assumed $\leq 1/f$).

Uniform Gossip uses a simple *infect and die* procedure. For each unique message it receives, the node will send it to f other nodes. If we consider only one source, and assume that most nodes are reached by the message, the number of messages exchanged in the system can be estimated as

$$|msg|_{uniform} \approx f \times |N|. \tag{1}$$

In the rest of our analysis, we assume, following [EGKM04], that the number of rounds needed by *Uniform Gossip* to infect a high proportion of nodes can be approximated by the following

Algorithm 5 – *Uniform Gossip* for a node (baseline)

(only showing main differences to Algorithm 4)

```
9′:  periodically Γ ← rpsViewSize nodes from RPS

15′: procedure GOSSIP(msg)
16′:   for all j ∈ {fanout random nodes in Γ} do
17′:     SENDTONETWORK(msg, j)

18′: upon receive (msg) do
19′:   if msg ∈ R then
20′:     return                    ▷ Infect and die: ignore if msg already received
21′:   R ← R ∪ {msg}
22′:   DELIVER(msg)                ▷ Deliver the message to the application
23′:   GOSSIP(msg)
```

expression when $N \to \infty$:

$$\lambda_{uniform} \approx \log_f(|N|) + C, \tag{2}$$

where C is a constant, independent of N.

GPS distinguished two subcategories of nodes: *Primary* nodes and *Secondary* nodes, noted P and S, that partition N. The density of *Primary* nodes, noted d, defines the size of both subsets:

$$|P| = d \times |N|, \qquad |S| = (1-d) \times |N|.$$

Primary nodes disseminate twice, while *Secondary* nodes disseminate once. Expressed differently, each node disseminates once, and *Primary* nodes disseminate once more. Applying the same estimation as for *Uniform Gossip* gives us:

$$\begin{aligned} |msg|_{GPS} &\approx f \times |N| + f \times |P| \\ &\approx f \times |N| + f \times d \times |N| \\ &\approx (1+d) \times f \times |N| \\ &\approx (1+d) \times |msg|_{uniform} \end{aligned} \tag{3}$$

(1) and (3) show that *GPS* only generates d times more messages than *Uniform Gossip*, with $d \in \mathbb{R}_{[0,1]}$. For instance, having 1% *Primary* nodes in the network means having only 1% more messages compared to *Uniform Gossip*.

If we now turn to the latency behavior of *GPS*, the latency of the *Primary* nodes is equivalent to that of *Uniform Gossip* executing on a sub-network composed only of $d \times |N|$ nodes, i.e.

$$\lambda_P \approx \log_f(d \times |N|) + C. \tag{4}$$

Combining (2) and (4), we conclude that *Primary* nodes gain $log_f(d)$ rounds compared to nodes in *Uniform Gossip*:

$$\Delta\lambda_P \approx -log_f(d). \tag{5}$$

Considering now *Secondary* nodes, their latency can be estimated a sum of three elements:

- the latency of *Primary* nodes;
- an extra round for *Primary* nodes to receive messages a second time;

- the latency of a *Uniform Gossip* among Secondary nodes with $d \times |N|$ nodes, corresponding to the *Primary* nodes, already infected;

which we approximate for $d \ll 1$ as

$$\begin{aligned}\lambda_S &\approx \log_f(d \times |N|) + 1 + \log_f(\frac{(1-d) \times |N|}{d \times |N|}) + 2C, \\ &\approx \log_f((1-d) \times |N|) + 1 + 2C. \end{aligned} \tag{6}$$

In summary, this analysis shows that *GPS* only generates a small number of additional messages, proportional to the density d of *Primary* nodes, and that the latency cost paid by *Secondary* is bounded by a constant value $(1+C)$ that is independent of the *Primary* density d.

3.4 Consistency Metric

Our second contribution is a novel metric that measures the consistency level of the intermediate states of an update-consistent execution. As discussed in Section 3.2, existing consistency metrics fall short either because they do not capture the ordering of operations, or because they cannot be computed without global system knowledge. Our novel metric satisfies both of these requirements.

3.4.1 A General Consistency Metric

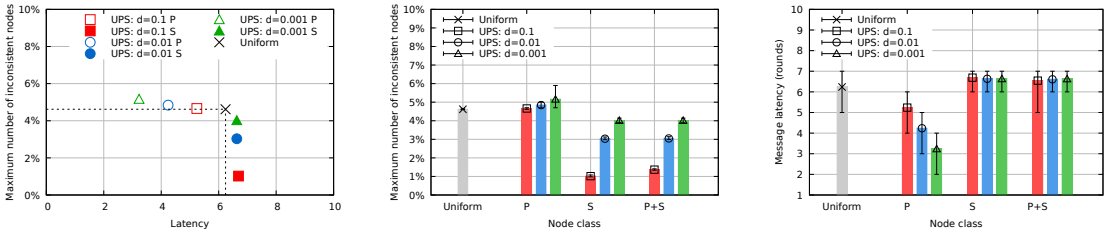
We start by observing that the algorithm for an update-consistent append-only message-queue that we introduced in Section 3.2 (Algorithm 3, page 16) guarantees that all its execution histories respect update consistency. To measure the consistency level of intermediate states, we therefore evaluate how the history deviates from a stronger consistency model, *sequential consistency* [Lam79]. An execution respects sequential consistency if it is equivalent to some sequential (i.e. totally ordered) execution that contains the same operations, and respects the sequential (process) order of each node.

Since update consistency relies itself on a total order, the gist of our metric consists in counting the number of read operations that do not conform with a total order of updates that leads to the final convergence state. Given one such total order, we may transform the execution into one that conforms with it by removing some read operations. In general, a data object may reach a given final convergence state by means of different possible total orders, and for each such total order we may have different sets of read operations whose removal makes the execution sequentially consistent. We thus count the level of inconsistency by taking the minimum over these two degrees of freedom: choice of the total order, and choice of the set[11].

More formally, we define a *transient inconsistency* of an execution Ex as a *finite* set of query events that, when removed from Ex, makes it sequentially consistent. We denote the set of all the transient inconsistencies of execution Ex over all compatible total orders by $TI(Ex)$. We then define the *relative inconsistency RI* of an execution Ex as the minimal number of query events that must be removed from Ex to make it sequentially consistent.

$$RI(Ex) = \begin{cases} \min\limits_{E \in TI(Ex)} |E| & \text{if } TI(Ex) \neq \emptyset \\ +\infty & \text{otherwise} \end{cases}$$

[11] With reference to Figure 19, if we consider the total order $< m.append(1), m.append(2), m.read(1), m.read(2), m.read(1,2), m.read(1,2) >$ then we need to remove both $m.read(1)$ and $m.read(2)$, while if we consider $< m.append(1), m.read(1), m.append(2), m.read(2), m.read(1,2), m.read(1,2) >$, we need to remove only $m.read(2)$.



(a) Experimental trade-off between consistency and latency (closer to the bottom left corner is better). A big gain on one side of the balance incurs a small penalty on the other side of the balance for a class.

(b) Consistency trade-off in *UPS* (lower is better). *Secondary* nodes are much more consistent than nodes in the baseline while *Primary* nodes only experience a small consistency penalty. The consistency of both *Primary* and *Secondary* nodes improves as the density of *Primary* nodes increases.

(c) Latency trade-off in *UPS* (lower is better). *Primary* nodes latency is lower than that of the baseline, the less *Primary* nodes, the lower their latency. *Secondary* nodes remain half a round slower than the baseline, regardless of the density of *Primary* nodes.

Figure 23: Consistency and latency trade-off in *UPS*. *Primary* nodes are faster and a bit less consistent, while *Secondary* are more consistent and a bit slower. The top of the bars is the mean while the ends of the error bars are the 5th and 95th percentiles.

For example, in Figure 19, removing $m.read(2)$ suffices to make the execution sequentially consistent. Therefore its relative inconsistency is 1.

The metric RI is particularly adapted to compare the consistency level of implementations of update consistency: the lower, the more consistent. In the best case scenario where Ex is sequentially consistent, $TI(Ex)$ is a singleton containing the empty set, resulting in $RI(Ex) = 0$. In the worst case scenario where the execution never converges (i.e. some nodes indefinitely read incompatible local states), every set of queries that needs to be removed to obtain a sequentially consistent execution is infinite. Since TI only contains finite sets of queries, $TI(Ex) = \emptyset$ and $RI(Ex) = +\infty$.

3.4.2 The Simpler Case of Append-Only Queues

In general, $RI(Ex)$ is complex to compute: it is necessary to consider all possible total orders of events that can fit for sequential consistency and all possible finite sets of queries to check whether there are transient inconsistencies. But in the case of an append-only queue implemented with Algorithm 3, we can easily show that there exists exactly one minimal set of transient inconsistencies.

To understand why, we first observe that the $append(x)$ is non-commutative. This implies that there exists a single total order of append operations that yields a given final convergence state. Second, Algorithm 3 guarantees that size of the successive sequences read by a node can only increase and that read operations always reflect the writes made on the same node. Consequently, in order to have a sequentially consistent execution, it is necessary and sufficient to remove all the query operations that return a sequence that is not a prefix of the sequence read after convergence. These read operations constitute the minimal set of transient inconsistencies TI_{min}.

3.5 Experimental Results

We perform the evaluation of *UPS* via PeerSim [MJ09], a well-known Peer-to-Peer simulator. A repository containing the code and the results is available on-line[**]. To assess the trade-offs between consistency levels and latency as well as the overhead of *UPS*, we focus the evaluation on three metrics:

[**] `https://gforge.inria.fr/projects/pgossip-exp/`

- the level of consistency of the replicated object;
- the latency of messages;
- the overhead in number of messages.

3.5.1 Methodology

Network Settings We use a network with 1 million nodes, a fanout of 10 and an RPS view size of 100. These parameters yield a broadcast reliability (i.e. the probability that a node receives a message) that is above 99.9%. Reliability could be further increased with a higher fanout [KMG03], but these parameters, because they are all powers of 10, make it convenient to understand our experimental results.

We use density values of *Primary* nodes of: $10^{-1}, 10^{-2}$ and 10^{-3}. According to Equation 4 in Section 3.3.4, we expect to see a latency gain for *Primary* nodes of 1, 2, and 3 rounds respectively. We evaluate four protocol configurations: one for *Uniform Gossip* (baseline), and three for *UPS* with the three above densities. then run each configuration 25 times and record the resulting distribution.

Scenario We consider a scenario where all nodes share an instance of an update-consistent append-only message queue, as defined in Section 3.4. Following the definition of update consistency, nodes converge into a strongly consistent state once they stop modifying the queue.

We opt for a scenario where 10 append(x) operations are performed on the queue by 10 random nodes, over the first 10 rounds of the simulations at the frequency of one update per round. In addition, all nodes repeatedly read their local copy of the queue in each round.

We expect the system to experience two periods: first, a transient situation during which updates are issued and disseminated (simulating a system continuously performing updates), followed by a stabilized state after updates have finished propagating and most nodes have converged to a strongly consistent state.

Consistency Metric Our experimental setting allows us to further refine the generic consistency metric we introduced in Section 3.4. In all our experiments, the number of update operations is finite and each node performs a query (read) operation during each round. For each round r, we define the sets $Q_P(r)$ and $Q_S(r)$ as the sets of all the query (aka *read*) operations performed at round r by the *Primary* and *Secondary* nodes, respectively.

This allows us to define a more precise metric to compare the evolution of the inconsistency of *Primary* and *Secondary* nodes through time. More precisely, $Incons_P(r)$ and $Incons_S(r)$ represent the proportion of *Primary* and *Secondary* nodes whose query are not correct at round r (i.e., the query is in TI_{min}).

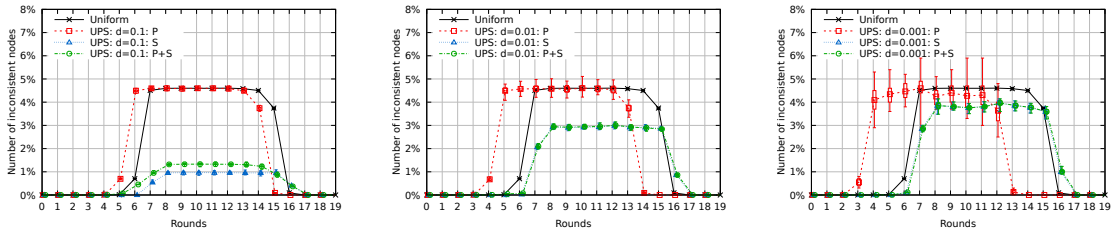
$$Incons_P(r) = \frac{|TI_{min} \cap Q_P(r)|}{d \cdot |N|}$$

$$Incons_S(r) = \frac{|TI_{min} \cap Q_S(r)|}{(1-d) \cdot |N|}$$

$$Incons_{P+S}(r) = d \cdot Incons_P(r) + (1-d) \cdot Incons_S(r)$$

As a result, the relative inconsistency of an experimental run Ex can be expressed as

$$RI(Ex) = |N| \cdot \sum_{r=0}^{\infty} (Incons_{P+S}(r)),$$



(a) $density = 10^{-1}$ (b) $density = 10^{-2}$ (c) $density = 10^{-3}$

Figure 24: While updates are being disseminated, *Secondary* nodes are more consistent than the baseline and *Primary* nodes are almost as consistent as the baseline. The more *Primary* nodes are in the system, the more *Secondary* nodes are consistent. Once updates are done being disseminated, *Primary* nodes converge faster to a consistent state.

and the metrics $Incons_X(r)$ discussed above can be interpreted as the proportion of inconsistent queries introduced in round r. For ease of exposition, we focus in the following on this *instantaneous* per-round measure, which is overall equivalent to $RI(Ex)$.

Plots Unless stated otherwise, plots use boxes and whiskers to represent the distribution of measures obtained over all runs for the represented metric. In the case of latency, the distribution is over all nodes within all runs. For inconsistency measures, the distribution is over all runs. The end of the boxes show the first and third quartiles, the end of the whiskers the minimum and maximum values, while the horizontal bar inside the boxes is the mean. In some cases, the low variance of the results makes it difficult to see the boxes and whiskers.

For clarity purposes, curves are slightly shifted to the right to avoid overlap between them. All the points between rounds r and $r+1$ belong to round r. In the following plots, *Primary* nodes are noted P, *Secondary* nodes are noted S and the system as a whole is noted $P+S$. Since only a small fraction of nodes are *Primary* nodes, the results of $P+S$ are naturally close to those of S.

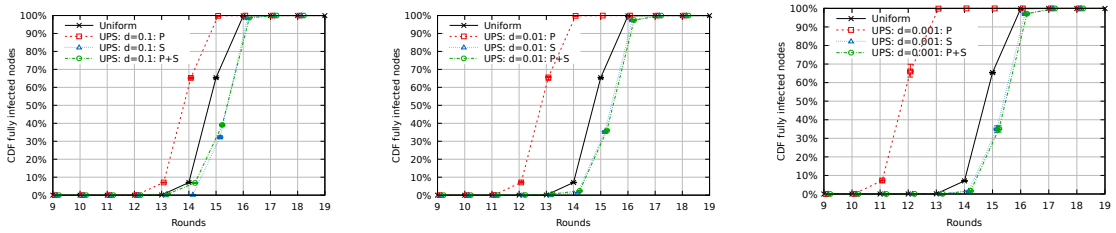
3.5.2 Overall results

Figure 23a mirrors Figure 20 discussed in Section 3.1 and provides an overview of our experimental results in terms of consistency/latency trade-off for the different groups of nodes involved in our scenario. Each *UPS* configuration is shown as a pair of points representing *Primary* and *Secondary* nodes: *Primary* nodes are depicted with hollow shapes, while *Secondary* nodes use solid symbols. *Uniform Gossip* is represented by a single black cross. The position on the x-axis charts the average update latency experienced by each group of nodes, and the y-axis their perceived level of inconsistency, taken as the maximum $Incons_X(r)$ value measured over all runs.

The figure clearly shows that *UPS* deliver the differentiated consistency/latency trade-offs we set out to achieve in our introduction: *Secondary* nodes enjoy higher consistency levels than they would in an uniform update-query consistency protocol, while paying only a small cost in terms of latency. The consistency boost strongly depends on d, while the cost in latency does not, reflecting our analysis of Section 3.3.4. *Primary* nodes present the reverse behavior, with the latency gains of *Primary* nodes evolving in the reverse direction of the consistency gains of *Secondary* nodes. We discuss both aspects in more details in the rest of this section.

3.5.3 Level of Consistency

Figure 23b details the consistency levels provided by *UPS* by showing the worst consistency that nodes experience over all simulations. We note an evident improvement of the maximum incon-



(a) $density = 10^{-1}$ (b) $density = 10^{-2}$ (c) $density = 10^{-3}$

Figure 25: *Primary* nodes receive all the updates faster than nodes in *Uniform Gossip*, they gain 1, 2 and 3 rounds for densities of $10^{-1}, 10^{-2}$ and 10^{-3} respectively. While *Secondary* nodes receive all the updates only half a round later on average compared to nodes in *Uniform Gossip*.

sistency level of *Secondary* nodes over the baseline and a slight decrease for *Primary* nodes. In addition, this figure clearly shows the impact of the density of *Primary* nodes over the consistency of *Secondary* nodes, and to a lesser extent over the consistency of *Primary* nodes.

Figures 24a, 24b and 24c show the evolution over time of the inconsistency measures $Incons_P(r)$, $Incons_S(r)$ and $Incons_{P+S}(r)$ defined in Section 3.5.1. We can observe an increase of inconsistencies during the transient phase for all configurations and a return to a consistent state once every node has received every update.

During the transient phase, the inconsistency level of *Primary* nodes is equivalent to that of nodes in *Uniform Gossip*. In that phase, around 4.6% of nodes are inconsistent with a higher variance at lower densities.

Meanwhile, the inconsistency level of *Secondary* nodes is much lower than that of nodes in *Uniform Gossip*. The density of *Primary* nodes plays a key role in this difference; the higher the density, the lower the inconsistency level of *Secondary* nodes. This level remains under 1.0% for the highest density but goes up to 4.0% for the lowest density.

The jitter, as defined in Section 3.3.2, is a good metric to compare the consistency of different sets of nodes: a lower jitter implies a higher consistency. The error bars in Figure 23c provide an approximation of the jitter of a set of nodes since they represent the bounds in latency of 90% of the nodes in the set. These error bars show that 90% of *Secondary* nodes receive updates within 1 round of margin, while the same proportion of *Primary* nodes and nodes in *Uniform Gossip* receive updates within 2 rounds of margin. This difference in jitter explains why *Secondary* nodes are more consistent than *Primary* nodes and nodes in *Uniform Gossip*.

Once the dissemination reaches a critical mass of *Primary* nodes, the infection of *Secondary* nodes occurs quickly. If the density of *Primary* nodes is high enough, then it becomes possible for a majority of *Secondary* nodes to receive the same update at the same time. Since it takes fewer rounds for *Secondary* nodes to be fully infected compared to *Primary* nodes, *Secondary* nodes turn out to be more consistent.

3.5.4 Messages Latency

Figure 23c represents the distribution of all the values of message latency over all the simulation runs. The three $P+S$ bars and the *Uniform* bar contain each 250 million message latency values (25 runs $\times$ 1 million nodes $\times$ 10 sources with a reliability of 99.9%).

This figure shows that *UPS* infects *Primary* nodes faster than *Uniform Gossip*. Specifically, *Primary* nodes obtain a latency gain of 1, 2 and 3 rounds with densities of 10^{-1}, 10^{-2} and 10^{-3} respectively. *Secondary* nodes, on the other hand, are infected half a round slower than nodes in *Uniform*

Gossip for all density values.

Figures 25a, 25b and 25c compare the evolution of the dissemination of updates between *Uniform Gossip* and *UPS* with different densities. Again, *Primary* nodes have a 1, 2 and 3 rounds speed advantage over the baseline while *Secondary* nodes are no more than a round slower.

We can also observe this effect on Figure 24 by looking at how fast all the nodes of a class return to a consistent state. We notice similar speed gain for *Primary* nodes and loss for *Secondary* nodes.

Overall, the simulation results match the analysis in Section 3.3.4 and confirm the speed advantage of *Primary* nodes over *Uniform Gossip* (Equation 5) and the small latency penalty of *Secondary* nodes (Equation 6).

3.5.5 Network Overhead

The number of messages exchanged in the simulated system confirms Equation 3 in Section 3.3.4. Considering the experienced reliability, we observe an increase in the number of messages of 10^{-1}, 10^{-2} and 10^{-3} compared to *Uniform Gossip* for all three densities of 10^{-1}, 10^{-2} and 10^{-3} respectively.

3.6 Related Work

UPS lies at the crossroad between differentiated consistency and gossip protocols (for the *GPS*-part). In the following, we review some of the most relevant works from these two areas.

Differentiated consistency A large number of works have looked at hybrid consistency conditions, originally for distributed shared memory [Fri95, AF96, KCZ92], and more recently in the context of geo-distributed systems [XSK+14, FRT15, LPC+12, TPK+13]. *Fisheye* [FRT15] and *RedBlue* [LPC+12] for instance both propose to implement hybrid conditions for geo-replicated systems. Fisheye consistency provides a generic approach in which nodes that are topologically close satisfy a strong consistency criterion, such as sequential consistency, while remote nodes satisfy a weaker one, such as causal consistency. This formal work focuses exclusively on immediate (i.e. non-eventual) consistency criteria and does not take convergence speed into account. RedBlue consistency offers a trade-off similar to that of *UPS*, but focuses on operations, rather than nodes, as we do. Blue operations are fast and eventually consistent while red operations are slow and strongly consistent.

Measuring Inconsistency Several papers have proposed metrics to evaluate a system's overall consistency. The approach of Zellag and Kemme [ZK12] detects inconsistencies in cloud services by finding cycles in a *dependency graph* composed of transactions (nodes) and conflicts between them (edges). Counting cycles in the dependency graph yields a measure of consistency that is formally grounded. It requires however a global knowledge of the system, which makes it difficult to use in practice at in large scale systems. Golab et al. introduced first Δ-atomicity [RGA+12, GLS11] and then Γ [GRA+14], two metrics that quantify data staleness in Lamport-atomic [Lam86] traces in the context of key-value stores. These metrics are not suitable for our problem since they do not take into account the ordering of update operations.

More practical works [LVA+15, PPR+11] evaluate consistency by relying on system specific information such as the similarity between different cache levels or the read-after-write latency (the first time a node reads the value that was last written).

Finally, CRDTs [SPBZ11] remove the need to measure consistency by only supporting operations that cannot create conflicts. This naturally leads to eventual consistency without additional ordering requirements on communication protocols.

Biased gossip protocols Many gossip broadcast protocols use biases to accommodate system heterogeneity. To the best of our knowledge, however, *GPS* is the first such protocol to target heterogeneous consistency requirements.

Directional gossip [LM99], for instance, favors weakly connected nodes in order to improve its overall reliability. It does not, however, target speed or consistency, as we do. The work in [CPOR07] looks at reducing a broadcast's message complexity by considering two classes of user-defined nodes: *good* and *bad* nodes. A new broadcast is disseminated to good nodes first using a reactive epidemic protocol, while bad nodes are reached through a slower periodic push procedure. As a result, the overall number of messages is reduced, at the cost of higher delivery latency for *bad* nodes. Similarly, *Gravitational gossip* [HJB+09] proposes a multicast protocol with differential reliability to better balance the communication workload between nodes, according to their capacities. Gravitational gossip associates each node with a susceptibility S_r and an infectivity I_r value that depend on a user-defined quality rating r. Nodes of rating r receive a fraction r of the messages before they time out. Gravitational gossip thus offers a cost/reliability trade-off, while *GPS* consider a consistency/latency trade-off. *Hierarchical gossip* [GKG06] also aims to reduce overheads but focuses on those associated with the physical network topology. To this end, it favors gossip targets that are close in the network hierarchy. This leads to a slight decrease in reliability and an increase of delivery latency.

In the context of video streaming, HEAP [FGK+09b] adapts the fanout of nodes to reduce delivery latency in the presence of heterogeneous bandwidth capabilities. In addition, nodes do not wait for late messages, but they simply ignore them. This dropping policy is well adapted to video streaming, cannot be applied to *GPS*. Finally, epidemic total order algorithms such as EpTO [MMF+15] and ecBroadcast [BGL+06] can be used to implement (probabilistic) strong consistency conditions, but at the cost of higher latency, and a higher number of messages for EpTO.

4 Conclusion

In this report, we have presented two contributions developed within the SOCIOPLUG project targeting the construction of emerging localities. The first contribution, SIMILITUDE, provides a decentralised overlay-based recommender that is able to adapt at runtime the similarity used by individual nodes. SIMILITUDE demonstrates the viability of decentralised locality adaptation for very large distributed systems, and shows it can compete against static schemes.

The second contribution, *Update-Query Consistency with Primaries and Secondaries* (*UPS*), is a novel eventual consistency mechanism that offers heterogeneous properties in terms of data consistency and delivery latency. *Primary* nodes can deliver updates faster, while *Secondary* nodes experience stronger data consistency at the expense of a small latency penalty. Both sets of nodes observe a consistent state with high probability once dissemination completes.

Both contributions illustrate the interest of systematic mechanisms for the construction of locality-based services in very large systems, and open promising avenues for future work. In particular, in the case of SIMILITUDE, 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.

Considering *UPS*, our future plans include deploying *UPS* in a real system and performing live experiments to confront the algorithm to real-life conditions. We also plan to investigate how *UPS* could be combined with a complementary anti-entropy protocol [DGH+87] to reach the last few susceptible nodes and further improve its performance.

References

[AAFE11] Ashraf M Attia, Nergis Aziz, Barry Friedman, and Mahdy F Elhusseiny. Commentary: The impact of social networking tools on political change in egypt's "revolution 2.0". *Electronic Commerce Research and Applications*, 2011.

[AEK00] Asim Ansari, Skander Essegaier, and Rajeev Kohli. Internet recommendation systems. *Journal of Marketing research*, 2000.

[AF96] Hagit Attiya and Roy Friedman. Limitations of fast consistency conditions for distributed shared memories. *Inf. Proc. Letters*, 57(5), 1996.

[ALR13] Sérgio Almeida, João Leitão, and Luís Rodrigues. Chainreaction: a causal+ consistent datastore based on chain replication. In *EuroSys*. ACM, 2013.

[AR12] Luca Maria Aiello and Giancarlo Ruffo. Lotusnet: tunable privacy for distributed online social network services. *Computer Communications*, 2012.

[BBG+] X. Bai, M. Bertier, R. Guerraoui, A.-M. Kermarrec, and V. Leroy. Gossiping personalized queries. In *EDBT'2010*.

[BD] Sonja Buchegger and Anwitaman Datta. A case for p2p infrastructure for social networks-opportunities & challenges. In *IEEE WONS 2009*.

[BDMR13] Ranieri Baraglia, Patrizio Dazzi, Matteo Mordacchini, and Laura Ricci. A peer-to-peer recommender system for self-emerging user communities based on gossip overlays. *J. of Comp. and Sys. Sciences*, 2013.

[BFG+a] Marin Bertier, Davide Frey, Rachid Guerraoui, Anne-Marie Kermarrec, and Vincent Leroy. The gossple anonymous social network. In *Middleware'2010*.

[BFG+b] Antoine Boutet, Davide Frey, Rachid Guerraoui, Arnaud Jégou, and Anne-Marie Kermarrec. Privacy-Preserving Distributed Collaborative Filtering. In *NETYS'2014*.

[BFG+13] A. Boutet, D. Frey, R. Guerraoui, A. Jégou, and A.-M. Kermarrec. WhatsUp Decentralized Instant News Recommender. In *IPDPS*, 2013.

[BGHS13] Peter Bailis, Ali Ghodsi, Joseph M Hellerstein, and Ion Stoica. Bolt-on causal consistency. In *ACM SIGMOD Int. Conf. on Man. of Data*, 2013.

[BGL+06] Roberto Baldoni, Rachid Guerraoui, Ron R. Levy, Vivien Quéma, and Sara Tucci Piergiovanni. Unconscious Eventual Consistency with Gossips. In *SSS*. 2006.

[BGYZ14] Sebastian Burckhardt, Alexey Gotsman, Hongseok Yang, and Marek Zawirski. Replicated data types: Specification, verification, optimality. *SIGPLAN Not.*, 49(1), 2014.

[BHG+] Ames Bielenberg, Lara Helm, Anthony Gentilucci, Dan Stefanescu, and Honggang Zhang. The growth of diaspora-a decentralized online social network in the wild. In *IEEE INFOCOM 2012 Comp. Comm. Workshops*.

[Bir07] Ken Birman. The Promise, and Limitations, of Gossip Protocols. *SIGOPS Oper. Syst. Rev.*, 41(5), 2007.

[Bre00] Eric A. Brewer. Towards robust distributed systems. In *PODC*, volume 7, 2000.

[BSVD] Sonja Buchegger, Doris Schiöberg, Le-Hung Vu, and Anwitaman Datta. Peerson: P2p social networking: early experiences and insights. In *SNS'2009*.

[CIK+] Jesús Carretero, Florin Isaila, A-M Kermarrec, François Taïani, and Juan M Tirado. Geology: Modular georecommendation in gossip-based social networks. In *ICDCS 2012*.

[CMS] Leucio Antonio Cutillo, Refik Molva, and Thorsten Strufe. Leveraging social links for trust and privacy in networks. In *iNetSec 2009*.

[CPOR07] N. Carvalho, J. Pereira, R. Oliveira, and L. Rodrigues. Emergent Structure in Unstructured Epidemic Multicast. In *DSN*, 2007.

[DDGR07] A. S. Das, M. Datar, A. Garg, and S. Rajaram. Google news personalization: scalable online collaborative filtering. In *WWW*, 2007.

[DGH+] Alan Demers, Dan Greene, Carl Hauser, Wes Irish, John Larson, Scott Shenker, Howard Sturgis, Dan Swinehart, and Doug Terry. Epidemic Algorithms for Replicated Database Maintenance. In *PODC'87*.

[DGH+87] Alan Demers, Dan Greene, Carl Hauser, Wes Irish, John Larson, Scott Shenker, Howard Sturgis, Dan Swinehart, and Doug Terry. Epidemic Algorithms for Replicated Database Maintenance. In *PODC*. ACM, 1987.

[dia] diaspora* statistics hub. `http://web.archive.org/web/20140320220658/http://pods.jasonrobinson.me/`. Accessed: 2014-03-20.

[DNDM] Elisabetta Di Nitto, Daniel J Dubois, and Alessandro Margara. Reconfiguration primitives for self-adapting overlays in distributed publish-subscribe systems. In *SASO 2012*.

[DS] Anwitaman Datta and Rajesh Sharma. Godisco: selective gossip based dissemination of information in social community based overlays. In *ICDCN 2011*.

[ea03] A. Ganesh et al. Peer-to-peer membership management for gossip-based protocols. *IEEE ToC*, 2003.

[EDPK09] M. El Dick, E. Pacitti, and B. Kemme. Flower-cdn: a hybrid p2p overlay for efficient query processing in cdn. In *EDBT*. ACM, 2009.

[EGH+03] P. Th. Eugster, R. Guerraoui, S. B. Handurukande, P. Kouznetsov, and A.-M. Kermarrec. Lightweight Probabilistic Broadcast. *ACM ToCS*, 21(4), November 2003.

[EGKM04] P Th Eugster, Rachid Guerraoui, A-M Kermarrec, and Laurent Massoulié. From epidemics to distributed computing. *IEEE computer*, 37(LPD-ARTICLE-2006-004):60–67, 2004.

[fac13] Facebook press statement: Q4 and full year 2012 results. `http://investor.fb.com/releasedetail.cfm?ReleaseID=736911`, 2013. Accessed: 2014-05-03.

[Fac14] Facebook Inc. Facebook: Company info – statistics. `https://newsroom.fb.com/company-info/`, March 2014. Accessed: 2014-05-13.

[FGK+09a] Davide Frey, Rachid Guerraoui, Anne-Marie Kermarrec, Boris Koldehofe, Martin Mogensen, Maxime Monod, and Vivien Quéma. Heterogeneous Gossip. In *Middleware*, 2009.

[FGK+09b] Davide Frey, Rachid Guerraoui, Anne-Marie Kermarrec, Boris Koldehofe, Martin Mogensen, Maxime Monod, and Vivien Quéma. Heterogeneous Gossip. In *Middleware*, 2009.

[FGK+09c] Davide Frey, Rachid Guerraoui, Anne-Marie Kermarrec, Maxime Monod, and Vivien Quéma. Stretching Gossip with Live Streaming. In *DSN*, 2009.

[FGK14] Davide Frey, Mathieu Goessens, and Anne-Marie Kermarrec. Behave: Behavioral cache for web content. In *DAIS*, 2014.

[FGKM10] Davide Frey, Rachid Guerraoui, Anne-Marie Kermarrec, and Maxime Monod. Boosting Gossip for Live Streaming. In *P2P*, 2010.

[FKM+] Davide Frey, Anne-Marie Kermarrec, Christopher Maddock, Andreas Mauthe, and François Taïani. Adaptation for the masses: Towards decentralized adaptation in large-scale p2p recommenders. In *13th Workshop on Adaptive & Reflective Middleware*, ARM '14.

[FKM+15] Davide Frey, Anne-Marie Kermarrec, Christopher Maddock, Andreas Mauthe, Pierre-Louis Roman, and François Taïani. Similitude: Decentralised adaptation in large-scale P2P recommenders. In Alysson Bessani and Sara Bouchenak, editors, *Distributed Applications and Interoperable Systems - 15th IFIP WG 6.1 International Conference, DAIS 2015, Held as Part of the 10th International Federated Conference on Distributed Computing Techniques, DisCoTec 2015, Grenoble, France, June 2-4, 2015, Proceedings*, volume 9038 of *Lecture Notes in Computer Science*, pages 51–65. Springer, 2015.

[FMP+16] Davide Frey, Achour Mostefaoui, Matthieu Perrin, Pierre-Louis Roman, and François Taïani. Speed for the elite, consistency for the masses: differentiating eventual consistency in large-scale distributed systems. In *to appear in the proceedings of the 35th Symposium on Reliable Distributed Systems*, Budapest, Hungary, September 2016.

[Fri95] Roy Friedman. Implementing hybrid consistency with high-level synchronization operations. *Dist. Comp.*, 9(3), 1995.

[FRT15] Roy Friedman, Michel Raynal, and François Taïani. Fisheye Consistency: Keeping Data in Synch in a Georeplicated World. In *NETYS*, 2015.

[gda] Google data centers, data center locations. `https://www.google.com/about/datacenters/inside/locations/index.html`. accessed Sep. 10 2015.

[GKG06] I. Gupta, A.-M. Kermarrec, and A.J. Ganesh. Efficient and adaptive epidemic-style protocols for reliable and scalable multicast. *IEEE TPDS*, 17(7), 2006.

[GL02] Seth Gilbert and Nancy Lynch. Brewer's conjecture and the feasibility of consistent, available, partition-tolerant web services. *ACM SIGACT News*, 33(2), 2002.

[GLS11] Wojciech Golab, Xiaozhou Li, and Mehul A. Shah. Analyzing consistency properties for fun and profit. In *PODC*, 2011.

[GNOT92] David Goldberg, David Nichols, Brian M Oki, and Douglas Terry. Using collaborative filtering to weave an information tapestry. *CACM*, 1992.

[GRA+14] Wojciech Golab, Muntasir Raihan Rahman, Alvin AuYoung, Kimberly Keeton, and Indarchand Gupta. Client-centric benchmarking of eventual consistency for cloud storage systems. In *ICDCS*. IEEE, 2014.

[HJB+09] K. Hopkinson, K. Jenkins, K. Birman, J. Thorp, G. Toussaint, and M. Parashar. Adaptive Gravitational Gossip: A Gossip-Based Communication Protocol with User-Selectable Rates. *IEEE TPDS*, 20(12), 2009.

[HOJ] István Hegedus, Róbert Ormándi, and Márk Jelasity. Gossip-based learning under drifting concepts in fully distributed networks. In *SASO 2012*.

[HXYS04] P. Han, B. Xie, F. Yang, and R. Shen. A scalable p2p recommender system based on distributed collaborative filtering. *Expert Systems with Applications*, 2004.

[IPKA10] T. Isdal, M. Piatek, A. Krishnamurthy, and T. Anderson. Privacy-preserving p2p data sharing with oneswarm. *SIGCOMM Computer Communication Review*, 2010.

[JMB09] Márk Jelasity, Alberto Montresor, and Ozalp Babaoglu. T-man: Gossip-based fast overlay topology construction. *Computer networks*, 53(13):2321–2339, 2009.

[JVG+07a] Márk Jelasity, Spyros Voulgaris, Rachid Guerraoui, Anne-Marie Kermarrec, and Maarten van Steen. Gossip-based peer sampling. *ACM TOCS*, 25, 2007.

[JVG+07b] Márk Jelasity, Spyros Voulgaris, Rachid Guerraoui, Anne-Marie Kermarrec, and Maarten Van Steen. Gossip-based peer sampling. *ACM ToCS*, 25(3), 2007.

[JVG+07c] Márk Jelasity, Spyros Voulgaris, Rachid Guerraoui, Anne-Marie Kermarrec, and Maarten Van Steen. Gossip-based peer sampling. *ACM ToCS*, 25(3):8, 2007.

[KCZ92] Pete Keleher, Alan L. Cox, and Willy Zwaenepoel. Lazy release consistency for software distributed shared memory. In *ISCA*. ACM, 1992.

[KLM] Irwin King, Michael R Lyu, and Hao Ma. Introduction to social recommendation. In *WWW 2010*.

[KLMT] Anne-Marie Kermarrec, Vincent Leroy, Afshin Moin, and Christopher Thraves. Application of random walks to decentralized recommender systems. In *OPODIS'10*.

[KMG03] A.-M. Kermarrec, L. Massoulie, and A.J. Ganesh. Probabilistic reliable dissemination in large-scale systems. *IEEE TPDS*, 14(3), 2003.

[KMM+97] Joseph A. Konstan, Bradley N. Miller, David Maltz, Jonathan L. Herlocker, Lee R. Gordon, and John Riedl. Grouplens: Applying collaborative filtering to usenet news. *CACM*, 1997.

[Kor] Yehuda Koren. Factorization meets the neighborhood: a multifaceted collaborative filtering model. In *ACM KDD'2008)*.

[KT] Anne-Marie Kermarrec and François Taïani. Diverging towards the common good: heterogeneous self-organisation in decentralised recommenders. In *SNS'2012*.

[KT+11] Anne-Marie Kermarrec, François Taïani, et al. Constellation: Programming decentralised social networks. 2011.

[Lam79] Leslie Lamport. How to make a multiprocessor computer that correctly executes multiprocess programs. *IEEE ToC*, 100(9), 1979.

[Lam86] Leslie Lamport. On interprocess communication. *Distributed Computing*, 1(2), 1986.

[LCB] V. Leroy, B. B. Cambazoglu, and F. Bonchi. Cold start link prediction. In *KDD'2010*.

[LFKA11] Wyatt Lloyd, Michael J Freedman, Michael Kaminsky, and David G Andersen. Don't settle for eventual: scalable causal consistency for wide-area storage with COPS. In *SOSP*. ACM, 2011.

[LM99] Meng-Jang Lin and Keith Marzullo. Directional Gossip: Gossip in a Wide Area Network. In *EDCC*. 1999.

[LPC+12] Cheng Li, Daniel Porto, Allen Clement, Johannes Gehrke, Nuno Preguiça, and Rodrigo Rodrigues. Making geo-replicated systems fast as possible, consistent when necessary. In *OSDI*, 2012.

[LPR07] J. Leitão, J. Pereira, and L. Rodrigues. HyParView: A Membership Protocol for Reliable Gossip-Based Broadcast. In *DSN*, 2007.

[LSY03] Greg Linden, Brent Smith, and Jeremy York. Amazon. com recommendations: Item-to-item collaborative filtering. *IEEE Internet Computing*, 2003.

[LVA+15] Haonan Lu, Kaushik Veeraraghavan, Philippe Ajoux, Jim Hunt, Yee Jiun Song, Wendy Tobagus, Sanjeev Kumar, and Wyatt Lloyd. Existential Consistency: Measuring and Understanding Consistency at Facebook. In *SOSP*. ACM, 2015.

[MCR11] Andres Moreno, Harold Castro, and Michel Riveill. Decentralized recommender systems for mobile advertisement. In *Workshop on Personalization in Mobile Applications (PEMA'11)*, Chicago, Illinois, USA, October 2011. ACM.

[MJ09] A. Montresor and M. Jelasity. PeerSim: A scalable P2p simulator. In *P2P*. IEEE, 2009.

[MKR04] B. N. Miller, J. A. Konstan, and J. Riedl. Pocketlens: toward a personal recommender system. *TOIS*, 2004.

[MMF+15] Miguel Matos, Hugues Mercier, Pascal Felber, Rui Oliveira, and José Pereira. EpTO: An Epidemic Total Order Algorithm for Large-Scale Distributed Systems. In *Middleware*. ACM, 2015.

[MMP] Giuliano Mega, Alberto Montresor, and Gian Pietro Picco. Efficient dissemination in decentralized social networks. In *IEEE P2P 2011*.

[mov] Movielens 1 million ratings dataset. `http://grouplens.org/datasets/movielens`.

[nei09] Time spent on facebook up 700 percent, but myspace.com still tops for video, according to nielsen. `www.nielsen.com/us/en/press-room/2009/time_on_facebook.html`, 2009. Accessed: 2014-05-03.

[nie] Nielsen social media report — q3 2011. `http://cn.nielsen.com/documents/Nielsen-Social-Media-Report_FINAL_090911.pdf`. Accessed: 2014-05-03.

[Ols] Tomas Olsson. Decentralised social filtering based on trust. In *AAAI-98 Recommender Systems Workshop*.

[PES+] J. M. Pujol, V. Erramilli, G. Siganos, X. Yang, N. Laoutaris, P. Chhabra, and P. Rodriguez. The little engine(s) that could: scaling online social networks. In *SIGCOMM 2010*.

[pin13] pingdom: Twitter.com (history). `http://stats.pingdom.com/wx4vra365911/23773/history`, 2013. Accessed: 2013-04-22.

[PMJ15] Matthieu Perrin, Achour Mostefaoui, and Claude Jard. Update Consistency for Wait-free Concurrent Objects. In *IPDPS*. IEEE, 2015.

[PPR+11] Swapnil Patil, Milo Polte, Kai Ren, Wittawat Tantisiriroj, Lin Xiao, Julio López, Garth Gibson, Adam Fuchs, and Billie Rinaldi. YCSB++: benchmarking and performance debugging advanced features in scalable table stores. In *Symp. on Cloud Comp. (SoCC)*. ACM, 2011.

[RGA+12] Muntasir Raihan Rahman, Wojciech Golab, Alvin AuYoung, Kimberly Keeton, and Jay J. Wylie. Toward a principled framework for benchmarking consistency. In *HotDep*, 2012.

[SD] Rajesh Sharma and Anwitaman Datta. Decentralized information dissemination in multidimensional semantic social overlays. In *ICDCN 2012*.

[SDP] Yading Song, Simon Dixon, and Marcus Pearce. A survey of music recommendation systems and future perspectives. In *CMMR'2012*.

[SNM] S. Scellato, A. Noulas, and C. Mascolo. Exploiting place features in link prediction on location-based social networks. In *ACM KDD'2011)*.

[SPBZ11] Marc Shapiro, Nuno Preguiça, Carlos Baquero, and Marek Zawirski. Conflict-Free Replicated Data Types. In *SSS*. 2011.

[SPMO02] A. Sousa, J. Pereira, F. Moura, and R. Oliveira. Optimistic total order in wide area networks. In *SRDS*. IEEE, 2002.

[SRF11] V. Schiavoni, E. Rivière, and P. Felber. Whisper: Middleware for confidential communication in large-scale networks. In *ICDCS 2011*, June 2011.

[SvSVV12] Vinay Setty, Maarten van Steen, Roman Vitenberg, and Spyros Voulgaris. PolderCast: Fast, Robust, and Scalable Architecture for P2p Topic-based Pub/Sub. In *Middleware*, 2012.

[THI+10] Juan M Tirado, Daniel Higuero, Florin Isaila, Jesús Carretero, and Adriana Iamnitchi. Affinity p2p: A self-organizing content-based locality-aware collaborative peer-to-peer network. *Comp. Net.*, 54, 2010.

[TLB] François Taïani, Shen Lin, and Gordon S. Blair. GossipKit: A unified component-framework for gossip. *IEEE TSE*, 40(2).

[TPK+13] Douglas B Terry, Vijayan Prabhakaran, Ramakrishna Kotla, Mahesh Balakrishnan, Marcos K Aguilera, and Hussam Abu-Libdeh. Consistency-based service level agreements for cloud storage. In *SOSP*. ACM, 2013.

[tri] Tribler. `http://www.tribler.org`.

[VN11] Steven Vaughan-Nichols. How skype does, and doesn't, work. `www.zdnet.com/blog/networking/how-skype-does-and-doesnt-work/1051`, 2011. Accessed: 2013-04-22.

[Vog09] Werner Vogels. Eventually Consistent. *CACM*, 52(1), January 2009.

[VPK+15] Abhishek Verma, Luis Pedrosa, Madhukar Korupolu, David Oppenheimer, Eric Tune, and John Wilkes. Large-scale cluster management at google with borg. In *EuroSys*. ACM, 2015.

[VS] S. Voulgaris and M. v. Steen. Epidemic-style management of semantic overlays for content-based searching. In *Euro-Par'05*.

[WUM10] Stéphane Weiss, Pascal Urso, and Pascal Molli. Logoot-undo: Distributed collaborative editing system on P2P networks. *IEEE TPDS*, 21(8), 2010.

[WZC+11] Dong Wei, Tao Zhou, Giulio Cimini, Pei Wu, Weiping Liu, and Yi-Cheng Zhang. Effective mechanism for social recommendation of news. *Physica A: Statistical Mechanics and its Applications*, 2011.

[XSK+14] Chao Xie, Chunzhi Su, Manos Kapritsos, Yang Wang, Navid Yaghmazadeh, Lorenzo Alvisi, and Prince Mahajan. Salt: Combining acid and base in a distributed database. In *OSDI*, 2014.

[YLL+09] Ching-man Au Yeung, Ilaria Liccardi, Kanghao Lu, Oshani Seneviratne, and Tim Berners-Lee. Decentralization: The future of online social networking. In *W3C Workshop on the Future of Social Networking*, 2009.

[Zie05] Cai-Nicolas Ziegler. *Towards decentralized recommender systems.* PhD thesis, Univ. of Freiburg, 2005.

[ZK12] Kamal Zellag and Bettina Kemme. How consistent is your cloud application? In *Symp. on Cloud Comp. (SoCC)*. ACM, 2012.