Constellation: Programming decentralised social networks

Anne-Marie Kermarrec[1]
[1] INRIA Rennes Bretagne Atlantique, Rennes, France
anne-marie.kermarrec@irisa.fr

François Taïani[1,2]
[2] Lancaster University, Lancaster LA1 4YW, UK
f.taiani@lancaster.ac.uk

1. Motivation

The past decade has witnessed a dramatic shift in the way the Web is used. The internet has entered our homes, our factories, and our lives like never before. User-generated content services (Flickr, Youtube, Delicious) and social networks (Twitter, FaceBook) have grown exponentially. The digital systems we live with are now composed of hundreds of millions of computing devices, of as many users, and of Terabytes of data, that is dynamically produced, shared, disseminated and searched globally. This data represents a fantastic potential to leverage information about each and every user (their circles of friends, their interests, their activities, the content they generate), and use it to provide better, more personalised, on-line services.

Although traditional social networks have been extremely successful, they today only encompass a tiny part of these many sources of information about users. It is not quite clear whether they should encompass more: Twitter has been known to experience regular outages because of the sheer load of data it must process. Facebook is regularly criticised out of privacy concerns.

These growing difficulties have prompted work on novel and fully decentralised alternatives to traditional social networks [2, 6]. Where traditional social engines fail to provide information that escape them, these decentralised data systems seek the information where it ultimately is: at the user.

To implement these decentralised social networks, gossip protocols appear as a natural solution, as they intrinsically tend to be highly resilient, efficient, and scalable. Existing gossip-based social networks have however so far been limited to simple use-cases, that typically take a uniform view of users, peers, and the data they hold [5, 1]. Similarly, the various mechanisms developed for search, recommendation, and personalisation in these systems currently only exist as independent standalone solutions, and lack a clear framework to allow their reuse, and composition, a key step to allow the incremental construction of more complex systems.

To progress to full-fledged decentralised social networks, we posit in this paper that we now need to move to gossip-based social applications that can simultaneously cater for different types of data and services. To design, and evaluate these new approaches in a modular and incremental manner, we further argue that we need specific and dedicated programming technologies. To help in this task, this paper sketches the main ingredients of a new programming language, Constellation, that seeks to simplify the realisation and experimentation with modular social gossip-based applications. Constellation is based on two central observations: (i) future decentralised social applications will need to handle heterogeneous forms of data and self-organisation, and (ii) these applications will need to better support modularity and incremental design, by providing appropriate mechanisms to compose data and topologies.

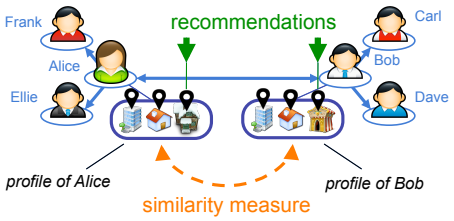

Figure 1: A typical gossip-based social network

2. Decentralised Social Networks

We first provide some background on decentralised social networks, before presenting the two main mechanisms of Constellation (Section 3.1).

2.1 System model

A traditional decentralised social networks uses a peer-to-peer architecture in which each peer (a smart phone, a tablet, a laptop) is associated with a user (Fig. 1). (We will discuss cases later in which a peer might also be associated to a web-site, a venue, a storage node.) Peers can connect to each other using point-to-point networking, but they only have a partial and very limited view of the rest of the system: Typically a small-size neighbourhood of other peers.

Each peer maintains some data (or profile) about its user, for instance her friends, or the set of items tagged in a collaborative tagging service (delicious), or the set of the locations visited in a geolocated social networks (foursquare). Peers use this data to organise themselves in a distributed overlay so that similar peers end up connected together. E.g. in Fig. 1, Alice and Bob have similar location profiles, and have therefore been selected to be neighbours of each other (using a mechanism we detail just below).

The resulting overlay can then be exploited to propose a range of personalised services such as search, recommendation, and query extension. The intuition behind this is that if Alice and Bob have similar profiles, then Alice's data can be leveraged to provide a better service to Bob. Here for instance, Alice might know of locations of interest to Bob (here the snowy house), and reciprocally (the circus tent).

2.2 Constructing the overlay

To construct its similarity overlay, a gossip-based social network uses a two-layer structure (Fig. 2) [4, 7]. Both layer maintain an overlay, in which peers have a fixed list of neighbours. In Fig. 2, Alice is connected to Bob, Carl, and Dave in the bottom RPS layer, and to Carl and Bob in the upper layer (clustering). Periodically, each peer selects a user from its view and exchanges information about its

neighbours with the final goal of converging to an optimal topology of users with 'similar' profiles in the top layer.

More precisely, the bottom RPS periodically provides each with a random sample of the rest of the network and thus guaranties the convergence of the second layer (clustering), while making the overall system highly resilient against churn and partitions. This is achieved by having peers exchange and shuffle their neighbours list in *periodic gossip rounds* to maximise the randomness of the RPS graph over time [3]. Alice might for instance request Carl's list of RPS neighbours, and randomly decide to replace Dave by Ellie (received from Carl) in her RPS view.

The clustering layer sits on top of the RPS layer, and implements a local greedy optimisation procedure that leverages both neighbours returned by the RPS, and current neighbours from the clustering views [4, 7]. A peer (say Alice in Fig. 2) will periodically update its list of similar neighbours with new neighbours found to be more 'similar' to her in the RPS layer. This guarantees convergence under stable conditions, but can be particularly slow in large systems. This mechanism therefore is complemented by a swap mechanism in the clustering layer (Fig. 3), whereby two neighbouring peers (here Alice and Bob) exchange their neighbours lists (Step 1), and seek to construct a better neighbourhood based on the other peer's information (Step 2).

In Fig. 3 for instance, Alice has a thing for hearts, and prefers like-minded people. Bob, on the other hand, has a minor interest in hearts, but is much more interested in diamonds. When Alice sends to Bob her current list of neighbours, and Bob sends Alice his (Step 1), both discover new potential neighbours closer to their interests. Alice thus drops Ellie for Carl, and Bob drops Alice for Ellie.

3. Constellation

3.1 Design goals

Constellation's two main design goals are *simplicity* and *composability*. This translates into mechanisms and programming constructs that can very concisely represent gossip-based social networks in which different types of peer and clustering approach coexist (Sec. 3.2), at different levels of composition (Sec. 3.3).

3.2 Heterogeneous self-organisation

The peer-to-peer system that results from the approach presented in Section 2 is similar to a large set of physical particles (the peers) submitted to a uniform *law of attraction* (the similarity measure). In this physical metaphor, the RPS layer plays the role as thermal excitation (as in simulated annealing), while the clustering layer provides a local gradient approach to converge to an optimal topology (i.e. one that maximise similarity between nodes).

In Constellation, that kind of uniform clustering system is described by indicating (i) which data each node holds (e.g. `interests` below, a simple list of strings denoting the tags used by a user); and (ii) which similarity measure to apply between nodes (with the keyword `clusters`; here counting the overlap in tags between two users).

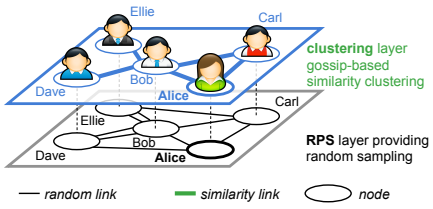

Figure 2: Gossip-based distributed clustering

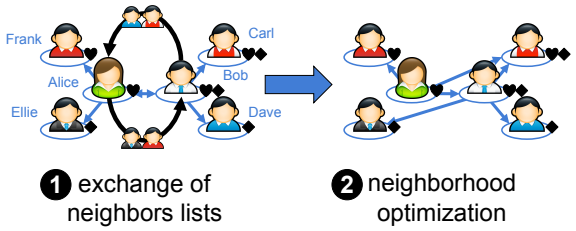

Figure 3: Peer-to-peer clustering mechanism

```
Website { List<String> topics }
User    {
  List<String> interests
  clusters potentialFriends { |u|
    overlap(this.interests,u.interest) }
  clusters interestingSites with Website { |site|
    cosSim( this.interests, site.topics ) }
  using potentialFriends.interestingSites }
```

Figure 4: Heterogeneous clustering

```
User { List<String> interests
       clusters potentialFriends { |u|
        overlap( this.interests, u.interests ) }}
```

Social applications may however involve different types of node (e.g. users and websites) and different similarities between these nodes, which a uniform approach cannot capture. To help program these more advanced systems, Constellation allows the declaration of multiple similarities between different types of nodes. For instance, in Fig. 4, `interestingSites` links each user with a list of websites this user might be interested in. With an RPS layer adapted to return both a random sample of both `User` and `Website` nodes, each user will eventually converge to a list of websites most likely to interest her. A complicating factor here comes from the fact that websites have no neighbourhood, which prevents the direct use of a gradient optimisation in the clustering layer (cf. Fig. 2). To work around this problem, Constellation provides the `using` keyword (last line in Fig. 4), which indicates here that user nodes should use their list of potential friends to optimise their list of available websites (a form of convergence piggybacking).

3.3 Virtualisation

The use of multiple similarity and heterogeneous peers described offer a world in which different 'attraction laws' (different similarities) can act on different particles (users, websites). In this model, all nodes are on the same level and usually correspond to a physical machine in a peer-to-peer system. Some social applications, however, can benefit from treating entities that are not associated with individual machines as 'virtually distributed', and thus provide an additional mechanism to compose data and topologies. This applies for instance to the search queries made by a user (in a decentralised search system), to the videos stored by a storage node (in a video storage system), or to the items

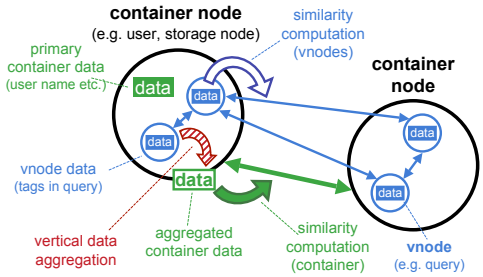

Figure 5: Vertical data aggregation

```
Query{ List<String> tags
       clusters similarQueries { |req|
        overlap(this.tags, req.tags) } }
User { contains Query queries
       clusters similarUsers { |u|
        overlap(this.queries.tags,u.queries.tags)}
       set scope similarUsers for queries }
```

Figure 6: Data aggregation & virtualised clustering

tagged by a contributor (in a collaborative tagging system).

To support these systems, CONSTELLATION allows nodes (called *containers*) to host "virtual nodes" (*vnodes* for short), i.e. entities which can maintain a similarity neighbourhood as nodes do (Fig. 5), but are not directly associated with a physical device. The neighbours of a vnode might belong to the same container node, or to distant containers. To allow vnodes to cluster themselves with other similar vnodes, CONSTELLATION includes two mechanisms: *vertical data-aggregation*, and *virtualised clustering*.

Vertical data aggregation allows a container node to access the data managed by its virtual nodes in the form of a multiset (a 'bag') of data items (red shaded arrow on Fig. 5). For instance, if a node `User` contains `Query` vnodes in a variable `queries` (Fig. 6), a user can access a collection of all her queries with `this.queries`. If each query maintains in turn a list of tags (as in Fig. 6), all tags contained in the queries of a user can be aggregated using `this.queries.tags`.

Using vertical aggregation, a CONSTELLATION program can define the neighbourhood of a container in terms of its vnode content (green solid arrow on Fig. 5). E.g. in Fig. 6 the similarity relationship `similarUsers` uses the list of tags of the queries made by a user to find similar users.

Virtualised clustering. Although vertical aggregation can be used on its own, its main use is to couple the similarity neighbourhood of vnodes with that of their containing node. This is useful semantically as vnodes (e.g. queries within users) are often contextually linked to their containing node: Similar queries from similar users are more likely to call for related answers, than similar queries from dissimilar users. Another reason pertains to efficiency: coupling the clustering of vnodes to that of containers allows for efficient realisations that do not overburden individual machines.

The general mechanism of virtualised clustering is shown in Fig. 7 for the code of Fig. 6: a vnode (e.g. a query **x**) can only maintain links with vnodes that lay in the neighbourhood of its containing node (here **Alice**). This is as if for **x**, the rest of the system only consists in **Alice** and her neighbours **Bob** and **Carl**, and the queries (vnodes) they contain (**u**, **v**, **w**, **y**, and **z**).

The clustering scope of queries is declared with the `set scope` keyword (Fig. 6), which indicates which similarity neighbourhood (here `similarUsers`) to use to scope the clustering of queries. The result in Fig. 7 is a two-level clustering system: users gravitate towards other users hosting similar queries, and queries are linked so similar searches in a user's neighbourhood, in a way similar to that proposed in [2].

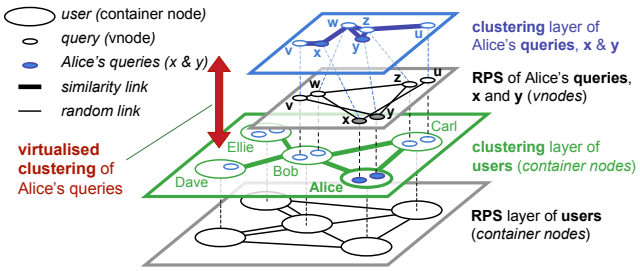

Figure 7: Virtualised clustering of Alice's queries (for the code of Fig. 6)

4. Outlook

CONSTELLATION offers two other features we have not discussed: the ability for nodes and vnodes to probe the data in their neighbourhood (*horizontal data aggregation*), and the capability of vnodes to migrate between containing nodes. Although functionally simple, these four mechanisms (virtualisation, virtualised clustering, data aggregation, and migration) can be combined arbitrarily, and we think offer a particularly attractive playing ground to experiment with novel gossip-based social algorithms.

CONSTELLATION suggests that social infrastructures might be approached in terms of bio-inspired systems, an aspect we would like to investigate. It might also provide paths to reason about the correctness a systems through static analysis, which we would like to explore in the future.

Acknowledgement

This work has been partially supported by the ERC Starting Grant GOSSPLE number 204742.

5. References

[1] X. Bai, M. Bertier, R. Guerraoui, A.-M. Kermarrec, and V. Leroy. Gossiping personalized queries. In *EDBT'10*.

[2] M. Bertier, D. Frey, R. Guerraoui, A.-M. Kermarrec, and V. Leroy. The gossple anonymous social network. In *Middleware'2010*.

[3] M. Jelasity, S. Voulgaris, R. Guerraoui, A.-M. Kermarrec, and M. van Steen. Gossip-based peer sampling. *ACM ToCS*, 25, August 2007.

[4] Márk Jelasity and Ozalp Babaoglu. T-man: Gossip-based overlay topology management. In *3rd Int. Workshop on Engineering Self-Organising App.*, 2005.

[5] V. Leroy, B. B. Cambazoglu, and F. Bonchi. Cold start link prediction. In *KDD '10*, 2010.

[6] 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*.

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