Constellation: Programming decentralised social networks

Anne-Marie Kermarrec, François Taïani

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

As they continue to grow, social and collaborative applications (e.g. twitter, facebook, digg) are increasingly calling for disruptive distributed solutions than can cater for the millions of users these applications serve daily, in hundreds of countries, over a wide variety of devices. To address these challenges, fully decentralised versions of social and collaborative applications are progressively emerging that seek to provide naturally scalable solutions to deliver their services. Gossip protocols in particular appear as a natural solution to implement these decentralised versions, as they intrinsically tend to be highly resilient, efficient, and scalable.

Social applications based on gossip have however been limited so far to relatively homogeneous systems: They typically rely on one similarity measure [3] to self-organise large amount of distributed users in implicit communities, and thus offer powerful means to search, mine, and serve personalised data in a distributed manner [1].

We posit in this paper that we now need to move to more complex gossip-based social applications that can cater for different types of data and similarity, organised in multiple levels of abstraction. Exploring, designing, and evaluating such novel approaches is unfortunately time-consuming and error-prone. 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 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) to offer more powerful services, these applications will need to move beyond physical nodes to encompass richer data structures organised in virtualised levels of abstractions.

2. Heterogeneous self-organisation

A traditional gossip-based social application [1] uses a two-layer structure to organise a large number of distributed nodes in similarity neighbourhoods (Fig. 1). Each node (e.g. representing a user) maintains a fixed list of neighbours in both layers, and periodically updates these view by interacting with its neighbours. In the top layer (*clustering*), nodes seek to construct a neighbourhood of

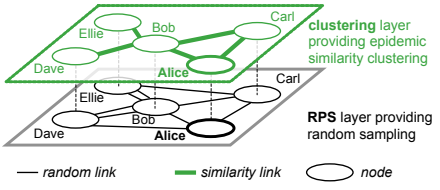

Figure 1: Gossip-based distributed clustering

```
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 2: Heterogeneous clustering

similar nodes (for instance nodes serving similar videos, or users with similar interest in their browsing history), according to some similarity measure (e.g. a count of interest tags in common) [3]. To that aim, they implement a local greedy optimisation procedure that leverages both the bottom random peer sampling (RPS) layer [2] (which guarantees convergence and provide resilience against churn and partition) and the clustering layer [3] (that speeds up convergence).

The resulting system is similar to a large set of physical particles 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).

```
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. 2, `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. 1). To work around this problem, Constellation provides the `using` keyword (last line

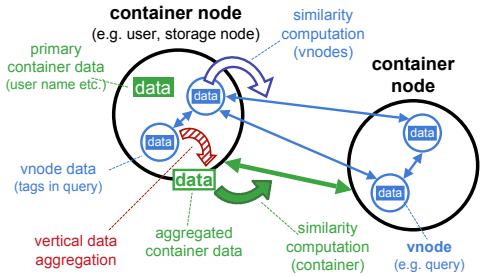

Figure 3: Vertical data aggregation

```
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 4: Data aggregation & virtualised clustering

in Fig. 2), 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. Virtualisation

The mechanisms just described offer a world in which different 'attraction laws' (different similarities) can act on different particles (different nodes). In this model, all nodes are on the same level and usually correspond to a physical a 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'. 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 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. 3), 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. 3). For instance, if a node `User` contains `Query` vnodes in a variable `queries` (Fig. 4), 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. 4), 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. 3). E.g. in Fig. 4 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 allow contained

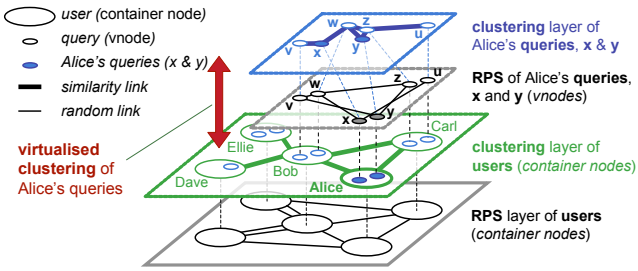

Figure 5: Virtualised clustering of Alice's queries (for the code of Fig. 4)

vnodes to create their own similarity neighbourhood while maintaining some coupling with 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 mechanics 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. 5 for the code of Fig. 4: 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. 4), which indicates which similarity neighbourhood (here `similarUsers`) to use to scope the clustering of queries. The result in Fig. 5 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 [1].

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.

In the mid term, we are exploring the possibility for vnodes to divide and merge (as cells would), which can be useful for storage and replication. CONSTELLATION might also offer paths to reason about the correctness and efficiency of a systems through static analysis, which we would like to further explore in the future.

5. References

[1] M. Bertier et al. The gossple anonymous social network. In *Middleware'2010*, pages 191–211.

[2] M. Jelasity et al. Gossip-based peer sampling. *ACM ToCS*, 25, 2007.

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