Designing extreme distributed systems: challenges and opportunities

[Extended Abstract]

Alberto Montresor
University of Trento, Italy
alberto.montresor@unitn.it

1. INTRODUCTION

Modern distributed systems may consist of hundreds of thousands of computers, ranging from high-end powerful machines to low-end resource-constrained wireless devices. We label them as *extreme distributed systems*, as they push scalability and complexity well beyond traditional scenarios.

Most of these systems are still organized along traditional lines with hierarchical, centralized control planes. Things are changing though: more and more decentralized organizations are emerging, exemplified by P2P systems, ad-hoc networks, vehicular networks, etc.

Decentralized organizations often combine local decision-making with dissemination of information in order to improve the decision-making process, exemplified by many gossip-based protocols [3]. These protocols have been designed to solve problems as diverse as information dissemination [4], aggregation [6], topology maintenance [7], heartbeat synchronization [1], etc. Solutions to these problems share many common aspects, yet they have been published and developed in a confused and scattered way, leaving developers alone when integrating them into real applications.

In this talk, we briefly introduce the gossip paradigm, showing how it can applied to solve several different problems, and we discuss what are the challenges that are left open for the researchers willing to provide a general framework for the design, implementation and deployment of gossip protocols.

2. THE GOSSIP PARADIGM

Since the seminal paper of Demers et al. [3], the idea of epidemic (or gossip) algorithms has gained considerable popularity within the distributed system community.

In a recent workshop on the future of gossip (summarized on a special issue of Operating System Review [8]), there has been a failed attempt to precisely define the concept of gossip. The reason for this failure is twofold: either the proposed definitions were too broad (including almost any message-based protocol ever conceived), or they were too strict (ruling out many interesting gossip solutions, some of them discussed here).

While a formal definition seems out of reach, it is possible to describe a prototypical gossip scheme that seems to entirely cover the intuition behind gossip. The scheme is presented in Figure 1.

```
1: loop
2:     wait($\Delta$)
3:     $q \leftarrow selectPeer()$
4:     $s_p \leftarrow prepareMsg(q)$
5:     send $\langle \text{REQ}, s_p \rangle$ to $q$
6: end loop
```

(a) active thread

```
1: loop
2:     receive $\langle t, s_q \rangle$ from $q$
3:     if $t = \text{REQ}$ then
4:         $s_q = prepareMsg(q)$
5:         send $\langle \text{REP}, s_q \rangle$ to $p$
6:     end if
7:     $update(s_q)$
8: end loop
```

(b) passive thread

Figure 1: Generic gossip scheme run by process p.

In this scheme, nodes regularly exchange information in periodic, pairwise interactions. The protocol can be modeled by means of two separate threads executed at each node: an active one that takes the initiative to communicate, and a passive one accepting incoming exchange requests.

In the active thread, a node periodically (every Δ time units) selects a peer node p from the system population through function *selectPeer*(); it extracts a summary of the local state through function *prepareMsg*(); and finally, it sends this summary to p. This set of operations is repeated forever. The other thread passively waits for incoming messages, replies in case of active requests, and modifies the local state through function *update*().

This scheme is still too generic and can be used to mimic protocols that are not gossip and thus must must be associated with a list of "rules of thumb" to distinguish gossip from non-gossip protocols:

- peer selection must be random, or at least guarantee enough peer diversity
- only local information is available at all nodes
- communication is round-based (periodic)
- transmission and processing capacity per round is limited
- all nodes run the same protocol

The main reason why gossip protocols are so interesting for implementing extreme distributed systems, is their robustness: node failures do not cause any major havoc to the system, and can be tolerated in large quantities; message losses often cause just a convergence speed reduction rather than safety issues. Low-cost is another plus: load is equally distributed among all nodes, in a way such that overhead may be reduced to few bytes per second per node.

The cause of such robustness and efficiency can be traced back to the inherently probabilistic nature of gossip protocols. They represent a "laid-back" approach, where individual nodes do not take much responsibility for the outcome. Nodes periodically perform a simple set of operations, they are not aware of the state of the entire system and act based on completely local knowledge. Yet, in a probabilistic sense, the system as a whole achieves very high levels of robustness to benign failures and a favorable (typically logarithmic) convergence time.

3. COMBINING GOSSIP PROTOCOLS

Beyond the original goal of information dissemination, gossip protocols have been used to solve a diverse collection of problems. More interestingly, it appears now that most of these solutions can be profitably combined to solve more complex problems. All together, these protocols start to look like a construction set, where protocols can be combined like Lego bricks. We show here a few examples.

Peer sampling provides each node with a continuously up-to-date sample of the entire population. This service solves a problem that lays at the basis of gossip: how to keep together the population of nodes that constitute the system, in such a way that it is possible implement function *selectPeer*() that selects nodes from such population. Instead of providing each node with a global view of the system, the peer sampling service provides each node with continuously up-to-date random samples of the entire population. Locally, each node only see the random node returned by *selectPeer*(); globally, the nodes and their samples define an *overlay topology*, i.e. a directed graph superimposed over the network. The graph is characterized by a *random* structure and the presence of a single strongly connected component.

Distributed aggregation is a common name for a set of functions that provide a summary of some global system property. In other words, they allow local access to global information in order to simplify the task of controlling, monitoring and optimizing distributed applications. Examples of aggregation functions include network size, total free storage, maximum load, top-k most frequent items, etc. Provided that the gossip partners are selected uniformly at random from the entire population of nodes through a peer sampling service, gossip-based aggregation protocols converge to the correct aggregate value, often in logarithmic time.

Load balancing. The aggregation protocol described above is *proactive*, meaning that all nodes participating in the computation are made aware of the final results. This suggests a simple improvement of a well-known load balancing protocol, as well as showing how simple protocol pieces can be combined together [5]. The load balancing scheme we want to improve works as follows: [2]: given a set of tasks that must be executed by a collection of nodes, nodes periodically exchange tasks in a gossip fashion, trying to balance the load in the same fashion as our average aggregation protocol. The problem with this approach is that tasks may be costly moved from one overloaded node to another overloaded node, without really improving the situation. Our idea is based on the concept that moving information about tasks is cheaper than moving tasks. For this reason, we use our aggregation service to compute the average load, and then later we put in contact – through a specialized peer sampling service – nodes that are underloaded with nodes that are overloaded. By avoiding overloaded-to-overloaded exchanges, this algorithm guarantees that an optimal number of transfers are performed.

4. CHALLENGES AND OPPORTUNITIES

Gossip protocols have been designed to solve the most diverse problems, and they started to be included in several systems – like in Amazon S3, to perform distributed monitoring, or in Bittorrent to diffuse chunks of data among peers. Yet, a more widespread adoption of this paradigm is hindered by the scattered and inconsistent state of research in this area.

While it has already been recognized that a *modular* approach for the development of gossip protocols is needed [5], we speculate that extreme distributed system designers should go even further and start adopting the *component-based software engineering* methodology, considering aspects of design, verification, testing, configuration and deployment.

Such approach would offer several attractive possibilities. It allows developers to plug different components implementing a desired function into existing or new applications, being certain that the function will be performed in a predictable and dependable manner. Components must be simple and predictable, as well as extremely scalable and robust. In this way, research can focus on self-organization and other important emergent features, without being burdened by the complexity of the protocols.

We believe that such multi-disciplinary effort could help gossip-based protocols to go outside their naive design and increase the adoption of this technology into mainstream applications.

5. REFERENCES

[1] O. Babaoglu, T. Binci, M. Jelasity, and A. Montresor. Firefly-inspired heartbeat synchronization in overlay networks. In *Proc. of SASO'07*. IEEE, 2007.

[2] A. Barak and A. Shiloh. A Distributed Load Balancing Policy for a Multicomputer. *Software Practice and Experience*, 15(9):901–913, Sept. 1985.

[3] A. Demers et al. Epidemic Algorithms for Replicated Database Management. In *Proc. of PODC'87*, 1987.

[4] P. T. Eugster, R. Guerraoui, S. B. Handurukande, P. Kouznetsov, and A.-M. Kermarrec. Lightweight probabilistic broadcast. *ACM Trans. Comput. Syst.*, 21(4):341–374, Nov. 2003.

[5] M. Jelasity, A. Montresor, and O. Babaoglu. A Modular Paradigm for Building Self-Organizing P2P Applications. In *Engineering Self-Organising Systems (LNCS 2977)*, pages 265–282. Springer-Verlag, 2004.

[6] M. Jelasity, A. Montresor, and O. Babaoglu. Gossip-based aggregation in large dynamic networks. *ACM Trans. Comput. Syst.*, 23(1):219–252, 2005.

[7] M. Jelasity, A. Montresor, and O. Babaoglu. T-Man: Gossip-based fast overlay topology construction. *Computer Networks*, 53(13):2321–2339, 2009.

[8] A.-M. Kermarrec and M. van Steen. Gossiping in distributed systems. *Operating Systems Review*, 41(5):2–7, 2007.