Think distributed: a glimpse of decentralized algorithmic [*]

Erwan Le Merrer & Gilles Trédan (INRIA Rennes, France)

1 Introduction

Since the birth of the computer era, the client-server paradigm has become the natural way of designing network architectures. As data and application logic are available at just one - or few - places (the servers), obvious simplicity follows concerning deployment, security or maintenance questions. Large mainframes and server farms are still nowadays at the core of all critical applications in industry or in public domain.

Only recently occurred one of the first original breakthrough w.r.t. this almighty monolithic paradigm; Napster and Freenet proposed around 1999 a new concept called *peer-to-peer*, were the client also plays the role of a server for the community of users. Those two states played simultaneously by users' computers relaxed the need for a powerful central server (in the case of Napster), or even removed the necessity of the server existence itself (case of Freenet). Many applications followed, from IP telephony, media streaming, virtual worlds, mail, to chat/messengers. Amongst properties that distribution allows, we can cite: *(i)* a very large number of participants (nowadays in millions), that centralized servers are unable to handle or at a prohibitive cost, *(ii)* fault tolerance and resilience to attacks (no single point of failure), *(iii)* anonymity (Freenet, ANTs P2P...), as the protocol's traffic do not transit through a single entity, and *(iiii)* computing resource exhaustive tasks, impossible to compute in reasonable time even by supercomputers (*e.g.* protein folding). Distributed systems include, along with peer-to-peer approaches, grid computing, sensor networks, and at a lower level multi-processors or distributed shared-memories.

At the application level, software algorithmic must be rethought. This is a direct consequence of the shift from a single "all seeing" point, to the distribution of program logic to a crowd of computers. The relative ease of algorithmic given by the fact that all data or client computers have a rendez vous at a particular place, is considerably complexified by the distribution of tasks.

This paper is interested in reviewing algorithms in some active research fields of purely distributed systems, that are algorithms that do not rely on a central network point, and were computers act collectively to perform a given task. The aim of this overview is of course not to be exhaustive, but instead to point out some interesting problematics, often still under heavy research in the distributed systems community. We believe those problematics will introduce to a non field familiar reader a new way of thinking protocols, compared to the traditionnal client-server approach.

This paper is structured as follows: Sections 1 and 2 introduce distribution, graph and complexity notions used in the rest of the paper. We then choose to divide distributed algorithmic into three categories of interests: individual, group, and global centric. We give some protocols for each category in Sections 4,5 and 6 before concluding.

2 What is distribution?

We are interested in this paper by what we call *fully decentralized/distributed algorithms.* Such approaches require the collaboration of at least a part of participants of a network, with the goal to achieve a computational task without the help of any centralized authority (server or global oracle). Based on their knowledge of the situation, all participants then need to act collectively, agree or take decisions to perform the target operation. The expression *emerging property* is often used to describe a global result that comes out of the sum local actions. Such paradigm generally supports the failure of some participants (disappearance of the network), loss of exchanged messages, and sometimes even non-conform/adversary behaviors. We review solutions for large-scale systems, *i.e.* of currently millions of participants (see *e.g.* Skype).

3 Graphs & complexity

Computer science relies on many mathematical abstractions. Trees, borrowed from graph theory, were for example heavily studied and used for sorting algorithms. Databases rely on set theory. Distributed systems are also often represented as

[*]This work is licensed under the Creative Commons Attribution 3.0 Unported License. Please contact authors for a copy or source code.

graphs, because this abstraction allows representing complex networks of interactions. Any social interaction, machine states or computer network can be represented as a *graph*, that depicts a "who knows who" relationship. An object (*node*) that has any kind of interaction with another one is virtually linked to it through an *edge*. Formally, the graph $\mathcal{G} = (V, E)$, is composed by the set of vertices V, and the set of edges E. n is the number of nodes in the graph.

Complexity theory is used in science to describe the amount of resources (time, messages, or space), needed by an algorithm to complete a given task. The *big O notation* is used in this context to describe the asymptotic behavior of algorithms: when the size of one parameter tends to infinity, one member of the problem's complexity equation then dominates; the other members are then skipped in the resulting notation, allowing easier comparison between different algorithms or approaches. $O(1)$ *e.g.* depicts a constant size problem, but $O(n^2)$ represents a complexity of the solution that is quadratic compared to the evolution of the input parameter n (here the size of the network). O symbol represents an upper bound, while Ω is the lower bound of a problem.

4 On the importance of individuals in networks

Graphs, as stated before, are commonly used to model networks of interactions. These interactions, sometimes also called dependencies, or collaborations, thus form a complex network whose analysis may be noteworthy. The underlying idea is that the efficiency of the phenomenon studied relies on the way the graph is knitted. For example, the efficiency, and moreover the resiliency of the power distribution network heavily relies on the way repartitors are interconnected and placed in the network. It is important for anybody interested in (or against) the continuity of power distribution to study this graph to identify the sensible parts composing it.

The difficulty of analysing large interaction graphs comes from the fact that they contain a huge amount of information. This amount of information needs to be filtered or aggregated to allow efficient analysis. To that end, physicians and sociologists introduced what is known as *centrality*. The centrality of a node in a given graph represents its importance w.r.t. the topology of this graph. Back to our power grid, the centrality of a given network part will for example be proportional to the number of people left out of power if this node happens to fail providing energy.

Since the notion of importance may take many forms, there are different forms of centralities. Here are some examples:

Degree centrality The perhaps simplest centrality is the *degree* centrality. A node's degree is the number of neighbors it has (*i.e.* the number of edges that end on the node). The idea is to say that a node is important for the graph if it has a lot of connexions. This is one of the metrics used to analyse terrorist networks [7].

Eigenvector centrality This is a refinement of the previous centrality. It aims to take into account the quality of the neighbors a node has. In other words, it says that an important node is a node that is connected to other important nodes. This is a basis of the algorithm used by Google to rank web pages.

Betweeness centrality This centrality belongs to a family of centralities that is different from the previous ones. The two previous centralities are related to degrees. This one refers to the path notion that we will quickly introduce hereafter. Formally a path is a set of consecutive edges in a graph. Imagine for example the graph representing the network of airports in the world: travelling from a medium-sized airport in France (Toulouse) to a city in the united states (Portland) cannot be done in one flight, and the traveller has to transit (for example) by Paris and Washington. From a graph theory point of view, this means that Toulouse and Portland are not neighboring vertices, and that a shortest path linking them is Toulouse-Paris-Washington-Portland. Additionally, we say that their distance (the minimal number of flights, or hops, to reach one from the other) is 3.

The betweeness centrality is related to the path notion: it represents the importance of a given node by the number of shortest paths it is involved in. It is interesting to notice that this centrality may be totally uncorellated to the previous ones: a node with a very small degree may be involved in a lot of paths. However, both centralities are often practically correlated (Paris' airports serve a lot of various destinations and so are involved on a lot of paths).

Attacking nodes with a high betweeness centrality enlarges the distances within the target graph, ultimately partitioning it. A nice application of this principle is fighting epidemics, for example during the avian flu: airports where hardly disinfected because they allowed the flu to travel fastly far away, thus creating "contamination shortest paths" all over the world. This way, contamination can only be done locally and thus the virus is spread slower.

These measures are of great interest for system architects. One of the qualities of a distributed system is the resiliency: it is harder to harm these kind of systems (the difficulties of governments in fighting illegal p2p file sharing is a good example) than to harm a single server. However, nodes having a too high centrality are critical for this resiliency property: many things rely on their existence.

Paper [5] presents a distributed algorithm to compute nodes importance. It uses a popular tool of mathematics, a *random walk*, that is a autonomous process that travels from a node to one of its neighbors (and so on) generally selected uniformly at random. Its principle is rather simple: a random walk perpetually travels the input graph. Nodes log the time elapsed between two consecutive visits of this random walk (*i.e.* named *return times*). Observing the standard deviation of these return times provides information on the underlying graph: important nodes will see the random walk more regularly (*i.e.* lower standard deviation) than poorly connected or placed nodes. Moreover if the graph is made of several sparsly connected components, the random walk get trapped in these parts, and the observed standard deviation is high. If the graph is very regular, the random walk travels easily all the graph, and the observed standard deviation is low, thus giving for each node, in a distributed way, a flavor of the graph health. Standard deviation produced on nodes can alternatively be analysed offline, to assess the criticity of particular nodes in a studied network of interactions.

5 Groups in graphs

In the previous section, we observed particular nodes in a given graph, capturing their importance. In this Section, we focus on identifying groups of nodes. Doing so is a non obvious and computationally expensive task. In a graph where node have a tendency to cluserize towards some interests, group identification hierarchically splits the graph into well delimited groups.

5.1 What is a group ?

There are many formal definitions of what is a "good" group, or a *cluster*. A classic approach is to define a partitioning $\{P_1, ..., P_p\}$ of the vertices ($\forall i, P_i \in V$), and to compare the number of internal, or *in-cluster* edges (edges that start and end in a given P_i) with the number of crossing, or *between-cluster* edges. If the number of internal edges is high compared to the number of crossing ones, then the graph exhibits a community structure that is well represented by this partitioning.

It is interesting to notice that here we do not impose any conditions on the relative size of partitions, nor on the number of such partitions. Closely related problems include the *graph partitioning* problem, that consists in finding the best split of a graph into two equally sized partitions such that the number of between-cluster edges is minimal.

5.2 Detecting communities

Distributed communities detection is a fairly recent problem. Like most of algorithms studying the structure of graphs, solutions are expensive since they often involve communication between any pair of nodes, which cost n^2 messages (for a fully connected graph). Recently, Pons and al. [11] proposed a distributed cluster detection algorithm based on random walks that runs in $O(n^2 \log(n))$ steps. Here is the underlying intuition: take a graph where vertices represent people and edges represent friendship relations. Suppose that this graph is clusterized (*i.e.* there are clusters of people that are often mutually friends and have few friends outside the cluster). Imagine they transmit an object (the random walk) such as a music disk. After a few friend-to-friend transmissions, say t, chances are high that the disk is still possessed by a people from the starting cluster: it has few chances to escape this community. In other words, the probability for a random walk to travel from a node i to a node j within t steps defines a *distance* between nodes i and j. This distance allows to distinguish between two nodes from a same cluster and two nodes from different clusters. Based on this, the algorithm then group the nodes that are close.

5.3 Applications

Exploiting groups in graphs may lead to several interesting possibilities. First of all, graphs exhibiting nice community structures are easier to handle and describe as soon as groups are named. Then, imagine for example the graph of Internet music listeners, where an edge links two users listening the same music (see *e.g.* [10]). The group structure is then something worth examining: they delimit music influences and styles, allowing, among other things, to represent both artists and listeners proximities. These informations may then be exploited for various goals: automatic *intelligent* mixing, content recommendation (a hot topic) and artist impact studies. From a network application point of view, such groups are very interesting since they delimit entities exhibiting the same behaviour.

6 And global properties?

On top of the pile, network could be considered as a whole so one can be interested in deriving global properties about that data mass. Since nearly two decades, distributed graph theory has studied algorithms over a tremendous panel of research fields. We overview in this section three application examples.

6.1 Decentralized Routing

One of the main question that is still under active research is "how do we navigate efficiently in a graph in a decentralized fashion?"; its simply asks how to get from a position (a node) to another one without the help of a central authority (*e.g.* a GPS!). One famous example that exhibits a surprising result about routing is the Milgram experiment, also known as the "six degrees of separation" experiment. The idea is the following: a random person is chosen in the USA, and is given a letter containing basic information about another target person (also in the USA). The first person is asked to participate by checking if its knows personally the target; if so it sends it directly the letter, so the experiment stops. Otherwise it forwards it to a contact that he thinks is the more likely to know the target. The metric is how many times the letter has been forwarded. It turns out that amongst the letters that arrived, the average path length was six hops. This somehow counter-intuitive claim, considering the size of the network composed by the American population, has been generalized and extensively studied.

This idea traduces one of the simplest routing strategy, the *greedy routing* method, were each node of a network forwards the message to its neighbor that is the closest to the final destination, thus based on pure local knowledge. Kleinberg [6] showed that augmenting a $n \times n$ two-dimensional grid with few random links (their probability of existence between each pair of nodes decreases has the distance increases) could provide a $O(\log^2 n)$ steps to destination for greedy routing. The trivial case on a grid without any random link is clearly $O(n)$. Researchers use now the *small world* expression to describe networks with short path lenghts. For other graph structuration and routing algorithms, please refer to this nice survey [9].

6.2 Epidemic protocols

Distributed systems currently make the use of *epidemics* (protocols that mimic the spread of a contagious disease [3]) for a lot of applications (*e.g.* from database update to clock synchronization). One of the more straightforward use is the propagation of an information, from a source node to the entire system. Considering an information, that each *infected* node pushes to f other nodes (randomly chosen) per round, it has been shown that the infection of the entire set of nodes takes a logarithmic number of steps ($\log_{f+1}(n) + \frac{1}{n}\log(n) + O(1)$).

A nature inspired epidemic protocol [1], copied from what happens at dusk for a specie of flashing fireflies, helps system designers to synchronize heartbeats of computers. Those are used to synchronize the start of execution cycles, or as failure detectors for example. A flash is seen here in our context as a message emission (*e.g.* simple UDP ping) from a node to all of its topology neighbors. Here is the sketch of an example algorithm to synchronize all nodes' flashes. Initially, all nodes start unsynchronized; they all own a variable cycle length, Δ, after which they send a flash. The goal is thus here to end up with the synchronization of the whole crowd of flashes. If a particular node receive a flash "too late" from one of its neighbor (that is at $t < \Delta/2$), Δ is lenghtened, so that next flash is more likely to be aligned. The exact opposite is processed when a flash is received "too early". Simulations show *e.g.* that for 2^{10} nodes, with their Δ randomly and uniformly distributed in $[0.82, 1.19]$ seconds, only a very few tens of seconds is necessary for nodes to emit time coherent flashes (here in a windows of 30ms).

6.3 Nodes' coverage

We here present the idea of interconnecting nodes together in an efficient fashion, and motivate it through the broadcast application. Nowadays, there is an increasing need for methods to efficiently disseminate information through messages from a node to all/a subset of other nodes. While epidemic based broadcast techniques, *e.g.* [4], provide a high level of resilience facing network dynamics or message loss, other classes of algorithm can serve as a basis to implement efficient dissemination with respect to the number of generated messages.

An interesting example is a class of algorithms that aims at creating a new structure amongst nodes, to help them to be reachable in an efficient way. *Spanning trees*, by linking nodes on a tree fashion, provide a deterministic way for nodes to propagate messages in a crowd (each father node in the tree forwards the message to its children); unfortunately, trees are hard to maintain in the presence of node arrivals/departures in large scale networks. In this context, *spanner graphs* are more resilient due to presence of cycles in the emerging structure. The idea is to remove communication links (edges) from the original structure, in order to provide in fine a sparse network, that will avoid massive redundancy in terms of messages used by broadcast techniques.

A recent result [2] provides an algorithm that constructs, in a constant (chosen) number of rounds, k, a spanner with optimal properties (w.r.t. the distributed paradigm). Removing edges in a graph have the consequence to increase distance between nodes (in terms of hops). The trade-off is thus here to obtain a structure with a small number of edges, while keeping a reasonable stretch ratio between the original distance and the distance in the created spanner. [2] insures a stretch of at most $2k + 1$ times the distance in the original graph, while completing in only k rounds and generating $O(kn^{(1+1/k)})$ edges; these values are known to be optimal considering the used model and the fact that the algorithm is deterministic (not using randomization). Here is a sketch of the algorithm performed at each node: each node owns a region (starting only with this particular node); all the regions put all together forms the spanner. Examine your neighbors (their region) at distance 1, include a particular node in your region if its region does not intersect any other regions that you known. In the reverse case, discard this node if its region overlaps one of a node in your region. Repeat by incrementing distance until distance equals k. The output region is the spanner.

7 Conclusion

As previewed in this short paper, the distributed paradigm imposes a new way of thinking protocols. Decentralization is of interest nowadays for a large field of applications; we chose to present a set high level questions, while reseachers are also designing and looking at the properties of chip-level distributed protocols [8]: as the Moore's law (doubling of the number of transistors on a chip every 18-24 months) ended, the aim to carry on improving computers' speed opened wide the doors for the multi-core age. The same problematic then arises: how to efficiently synchronize, give mutual exclusion primitives, to a set of independently acting processors? We believe that the understanding and the ongoing developpment of distributed systems and algorithmic is of major importance for the future of computer science.

References

[1] O. Babaoglu, T. Binci, M. Jelasity, and A. Montresor. Firefly-inspired heartbeat synchronization in overlay networks. In *SASO '07: Proceedings of the First International Conference on Self-Adaptive and Self-Organizing Systems*, pages 77–86, Washington, DC, USA, 2007. IEEE Computer Society.

[2] B. Derbel, C. Gavoille, D. Peleg, and L. Viennot. On the locality of distributed sparse spanner construction. In *PODC '08: Proceedings of the twenty-seventh ACM symposium on Principles of distributed computing*, pages 273–282, New York, NY, USA, 2008. ACM.

[3] P. Eugster, R. Guerraoui, A.-M. Kermarrec, and L. Massoulie. Epidemic information dissemination in distributed systems. *Computer*, 37(5):60–67, May 2004.

[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, 2003.

[5] A.-M. Kermarrec, E. Le Merrer, B. Sericola, and G. Trédan. Evaluating the quality of a network topology through random walks. In *DISC*, pages 509–511, 2008.

[6] J. Kleinberg. The small-world phenomenon: An algorithmic perspective. In *in Proceedings of the 32nd ACM Symposium on Theory of Computing*, pages 163–170, 2000.

[7] V. E. Krebs. 2002 insna mapping networks of terrorist cells.

[8] J. Larus and C. Kozyrakis. Transactional memory. *Commun. ACM*, 51(7):80–88, 2008.

[9] K. Lua, J. Crowcroft, M. Pias, R. Sharma, and S. Lim. A survey and comparison of peer-to-peer overlay network schemes. *Communications Surveys & Tutorials, IEEE*, pages 72–93, 2005.

[10] J. C. Platt. Fast embedding of sparse music similarity graphs. In *In Advances in Neural Information Processing Systems*, page 2004, 2004.

[11] P. Pons and M. Latapy. Computing communities in large networks using random walks (long version), 2005.