Self-organizing Systems Case Study: peer-to-peer systems*

Emmanuelle Anceaume, Maria Gradinariu, and Matthieu Roy

IRISA, Campus de Beaulieu, 35042 Rennes, France
{anceaume, mgradina, mroy}@irisa.fr

Abstract. This paper aims at providing a definition of self-organization for dynamic and open systems such as peer-to-peer networks. We propose a definition of self-organization based on the locality principle — a node maintains only a limited amount of information with respect to a bounded set of nearby nodes. As opposed to traditional peer-to-peer systems that focus on data placement and request routing, we present a dual approach which reorganizes links and spontaneously groups nodes using similarity criteria, thus extending the locality principle to the network topology. Moreover we propose an architectural model of the peer-to-peer networks relying on the self-organization of the system in multiple layers, each of them reflecting the self-organization with respect to a particular evaluation criterion (e.g. connectivity, load balancing, data similarity ...). This architecture is adapted to the implementation of a scalable set of services. We propose, as an example, an implementation of a data tracking service.

1 Introduction

The major feature of all recent scalable systems is their extreme dynamism in structure, content and load [17]: nodes are continuously joining and leaving the system, there is no central entity in charge of their organization and control, and there is an equal capability, and responsibility entrusted to each of them to own data [10]. To cope with such characteristics, these systems have to be able to spontaneously organize toward desirable global properties. Different kinds of self-organization can be addressed. In peer-to-peer systems, self-organization is handled through protocols for node arrivals and departures based on a fault-tolerant overlay network as in Pastry [7, 16], or through localization and routing infrastructure as in OceanStore [11, 20]. In ad hoc networks, solutions have been proposed for a self-organizing public-key management system that allows users to create, store, distribute, and revoke their public keys without the help of any trusted authority or fixed server [5]. The idea of self-organization or related variants as self-configuration, self-healing or reconfiguration was studied in [19] or [18]. The former work proposes the concepts of self-healing and self-configuration in wireless networks, while the latter one focuses on the concept of reconfiguration of a metamorphic robotic system with regard to a goal configuration. Finally, one could argue that the self-stabilization framework [4, 6] specifies the self-organization of highly dynamic networks. Actually, the nature of the dynamic networks cannot fit the self-stabilization process essentially because self-stabilization deals with systems affected by infrequent and transient faults, which make them capable to converge toward a stable pre-defined configuration. This is orthogonal to self-organizing systems which should be relatively insensitive to perturbations or faults, and should have a strong capacity to restore themselves, essentially because these systems thrive on fluctuation or "noise".

To be confident in the capability of a system to be self-organized, one needs to rely on a well-accepted notion of self-organization. The first contribution of this paper is to propose a formal specification of this notion which, for the best of our knowledge, has never been done before in the area of scalable and dynamic systems despite a non negligible use of this term. This specification is based on the principles that govern dynamic systems. The first one concerns the exchange of information or resources with the environment (components are possibly capable to infinitely often retrieve new information/resources from components around them). The second one is the dynamics of these systems (components have the ability to move around, to leave or to join these systems based on local knowledge). The third principle is the specificity of the components: Among all components of the system, some have huge computation resources, some

* This work is supported by France Telecom R&D, SPEERAL contract nr. 423 38 322

have large memory space, some are highly dynamic, some have broad centers of interest. By looking at these systems as a mass of components, we obviate any differences that might exist between individual components, differences which make the richness of these systems.

All these tenets have as common seed the locality principle, that is some range of effect, both in terms of interaction and knowledge. We formalize this idea, leading first to the notion of *local self-organization*. Intuitively, a locally self-organizing system should force components to be adjacent to components that improve or at least maintain some property or evaluation criterion. By imposing the system to be locally self-organizing at all its nodes and by ensuring that despite the frequent and unpredictable connections/disconnections, the knowledge within the system progressively enriches, we then formalize the notion of *self-organization*.

The second contribution of this work is a case study. We consider the location data problem in peer-to-peer systems. Two main approaches have been proposed to tackle this challenging problem, the flooding approach and the distributed hash table (DHT) approach. The former one, adopted in Freenet [8], mainly consists for a node in sending a query to its neighbors which act similarly, until a time to live (TTL) query parameter is reached. Clearly, the dependability[1] of this approach is a function of the value given to the TTL parameter (a too small value may lead a part of the system uncovered, while a too large one makes the system unnecessarily congested). The DHT approach, whose main representative are CAN [14], Pastry [15], Chord [17], and Tapestry [20] is based on a deterministic mapping between keys of data and addresses of nodes. Such an approach enables a guaranteed and efficient localization of keyed data items, however it has the problem that the document IDs must be known before posting a request for a given document. We propose a different approach which exploits the self-organization property of the system to advance our performance objective of minimizing data localization latency. We build the data localization service on top of a multi-criteria self-organization layer. This layer is made up of three sub-layers, each of them dynamically organizing the topology of the system according to a given criteria. Such an approach enables nodes to decide whom to contact and when to add or drop a connection based on local knowledge only. By resorting on this layered organization, the retrieval of a data having the maximal similarity with the data owned by a given node needs $O(1)$ computation steps.

The remaining of this paper is organized as follows: Section 2 proposes a model for dynamic and scalable systems. Section 3 formalizes the local and global self-organization properties. In Section 4, an architectural model for peer-to-peer systems relying on the self-organization of the system in multiple layers is presented. Section 5 concludes and discusses open issues.

2 Model

2.1 Communication Graph

The network is described by a weakly connected graph. Its nodes represent processes of the system and its edges represent established communication links between processes. The graph is referred in the following as the communication graph. We assume that the communication graph is subject to frequent and unpredictable changes: processes can leave or join the system arbitrarily often, and they can fail temporarily (transient faults) or permanently (crash failures). Communication links can commit transient failures (e.g. messages loss). Notice that skip graphs [2] are possible candidate for the modeling of these systems, however they do not exploit geographical proximity.

Furthermore, we consider that the network is represented by a logical multi-layer system, each logical layer l being a weakly connected graph, also referred to as the communication graph at layer l. In order to connect to a particular layer l, a process executes an underlying connection protocol. A process p is called *active* at a layer l if there exists at least one process q which is connected at l and aware of p. The set of logical *neighbors* of a process p at a layer l is the set of processes q such that the logical link (p, q) is up (p and q are aware of each other). Note that a process i may belong to several layers simultaneously. Thus, i may have different sets of neighbors at different logical layers.

2.2 State Machine-based Framework

To describe and analyze such distributed and dynamic system rigorously, we use the dynamic I/O automata introduced by Attie and Lynch [3]. This model allows the modeling of individual components, their

[1] The intuitive notion of dependability for these systems is the reachability of information [12]

interactions and their changes. The external actions of a dynamic I/O automata are classified in two categories, namely the input-output actions (I/O actions), and dynamic actions (C/D actions for Connection-Disconnection actions) describing the mobility within the system. We introduce the notion of configuration which is defined as the system state at time t altogether with the communication graph. Note that this notion differs from the one of Attie and Lynch [3] which refers to the notion of global state. An execution $e = (c_0, \ldots, c_t, \ldots)$ of a dynamic I/O automaton is an infinite sequence of configurations, where c_0 is the initial configuration of the system. Every transition $c_i \rightarrow c_{i+1}$ is associated to the execution of one of the previously defined actions. Since the system is continuously evolving, the communication graph may change as the result of the execution of C/D actions. Let e be an execution of a dynamic I/O automaton. A *fragment* of e is a finite subsequence of e. Its size is the subsequence length. A *static fragment* is a maximal-sized fragment that does not contain C/D actions.

Let $f = (c_i, \ldots, c_j)$ be a fragment in a dynamic I/O automaton execution. We denote as begin(f) and end(f) the configurations c_i and c_j respectively. In the sequel, all the referred fragments are static and are denoted by f or f_i. Thus, an execution e of a dynamic I/O automaton is a infinite sequence of fragments: $e = (f_0, \ldots, f_i, \ldots f_j, \ldots)$.

3 Property of Self-Organization

In this section we propose to formally define self-organization in the context of scalable and dynamic systems. This specification, which strongly relies on the local self-organization property (see Section 3.1), is characterized by liveness and safety properties (see Section 3.2).

3.1 Local self-organization

Intuitively, a locally self-organizing system should force processes to be adjacent to processes which improve or at least maintain some evaluation criterion. An evaluation criterion $\mathcal{C}$ is either semantic-based (e.g., topic), or topological-based (e.g., load balancing). Let $\mathcal{C}$ be $[0,1]$-valued function defined on the local neighborhood of a process (the local neighborhood of a process p includes both the state of p and the state of p's neighbors). As we shall see, we associate to every criterion $\mathcal{C}$ a logical layer of the network. In the following $\mathcal{C}$ denotes an evaluation criterion, $\mathcal{C}_p(r)$ the evaluation of $\mathcal{C}$ by process p for process r, and $\mathcal{N}^{\mathcal{C}}(p)$ the set of p's logical neighbors for $\mathcal{C}$. This function is assumed to be locally computable.

In order to define local self organization, we introduce the notion of *stable configurations*. Informally, a configuration is *p-stable* for a given evaluation criterion $\mathcal{C}$ if, for any neighbor q of p, there is no neighbor r of q ($r \neq p$) that improves $\mathcal{C}$.

Definition 1 (p-stable configuration). *Let c be a configuration of a system $\mathcal{S}$ and p be a process. Configuration c is p-stable for the evaluation criterion $\mathcal{C}$ if* $\forall q \in \mathcal{N}^{\mathcal{C}}(p), \forall r \in \mathcal{N}^{\mathcal{C}}(q),\ \mathcal{C}_p(r) \leq \mathcal{C}_p(q)$

Definition 2 (local self-organization). *Let $\mathcal{S}$ be a system, p a process, and $\mathcal{C}$ an evaluation criterion. $\mathcal{S}$ is locally self-organizing for $\mathcal{C}$ in the neighborhood of p if $\mathcal{S}$ converges to a p-stable configuration.*

Module 1 proposes a self-organizing algorithm for a generic criterion $\mathcal{C}$ referred in the following as LSA. This algorithm is based on a greedy technique, which reveals to be a well adapted technique for function optimization. Its principle follows the here above intuition: Let q such that $q \in \mathcal{N}^{\mathcal{C}}(p)$, and r such that $r \in \mathcal{N}^{\mathcal{C}}(q)$ but $r \notin \mathcal{N}^{\mathcal{C}}(p)$. If p notices that r improves the evaluation criterion previously computed for q, then p replaces q by r in $\mathcal{N}^{\mathcal{C}}(p)$. Inputs of this algorithm are the evaluation criterion $\mathcal{C}$ and the set of p's neighbors for $\mathcal{C}$, that is $\mathcal{N}^{\mathcal{C}}(p)$. The output is the updated view of $\mathcal{N}^{\mathcal{C}}(p)$. Note that because of the partial view that a component has about the global state of the system (due to the scalability and dynamism of the system) only an heuristic algorithm can be found under these assumptions, i.e., one cannot expect to find an optimal one.

Theorem 1. *Let $\mathcal{S}$ be a system. If $\mathcal{S}$ executes the LSA algorithm with the evaluation criterion $\mathcal{C}$, then $\mathcal{S}$ is a locally self-organizing system for $\mathcal{C}$.*

For space limitations, proof of the theorem is given in [1].

Module 1 Local Self-Organization Algorithm for Criteria $\mathcal{C}$ Executed by p (LSA)

Inputs :

$\mathcal{C}_p$: the evaluation criterion used by p;
$\mathcal{N}^{\mathcal{C}}(p)$: p neighbors for the evaluation criterion $\mathcal{C}$;

Actions :

$\mathcal{R}$: if $\exists q \in \mathcal{N}^{\mathcal{C}}(p), \exists r \in \mathcal{N}^{\mathcal{C}}(q), \mathcal{C}_p(q) < \mathcal{C}_p(r)$
then $\mathcal{N}^{\mathcal{C}}(p) = \mathcal{N}^{\mathcal{C}}(p) \bigcup \{r_{max}\} \setminus q$;
where $r_{max} \in \mathcal{N}^{\mathcal{C}}(q)$, $\mathcal{C}_p(r_{max}) = max_{r' \in \mathcal{N}^{\mathcal{C}}(q), \mathcal{C}_p(q) < \mathcal{C}_p(r')}(\mathcal{C}_p(r'))$

3.2 Self-Organization through Local Self-Organization

As previously said, self-organization strongly relies on the local self-organization property, and on the behavior of the system during the connection/disconnection actions. According to this behavior, the system guarantees different levels of self-organization, namely, the weak and the strong self-organization. Before defining these properties, we introduce the notion of *global evaluation criterion*. The global evaluation criterion evaluates the global knowledge of the system at a given configuration. For instance, if the evaluation criterion $\mathcal{C}$ is proximity (i.e., the closer a process, the higher the evaluation criterion), then optimizing the global evaluation criterion γ will result in each process being connected to nearby processes. In the sequel we focus only on the global evaluation criterion which could be expressed as the union/intersection of local criteria. That is, γ is the sum of the local criteria:

$$\gamma(c) = \sum_{p \in \mathcal{S}} \sum_{r \in \mathcal{N}^{\mathcal{C}}(p)} (\mathcal{C}_p(r))$$

with p and r two processes of the system $\mathcal{S}$, and $\mathcal{C}_p(r)$ the evaluation by p of $\mathcal{C}$ in the configuration c.

The weak self-organization is defined in terms of two properties. The first one says that either infinitely often, there are static fragments during which the knowledge of the system enriches, or all processes have reached a stable state (*liveness* property), while the second one (*weak safety* property) states that during all static fragments, system knowledge does not decrease. Formally, we have:

Definition 3 (Weak Self-Organization). *Let $\mathcal{S}$ be a system and γ be an evaluation criterion defined on the configurations of $\mathcal{S}$. A system is* weakly self-organizing *for γ if the following two properties hold (recall that fragments stand for static fragments):*

liveness property:

$$\forall e = (f_0, \ldots, f_i, \ldots, f_j, \ldots), \forall f_i \in e, \exists f_j \in e, j \geq i : \quad \gamma(end(f_j)) > \gamma(begin(f_j)) \text{ or } \forall p \in \mathcal{S}, begin(f_j) \text{ is } p - stable$$

weak safety property:

$$\forall e = (f_0, \ldots, f, \ldots), \forall f \in e : \gamma(end(f)) \geq \gamma(begin(f))$$

The weak self-organization definition applies to static fragments. Nothing is guaranteed during non-static ones, that is fragments in which connections/disconnections occur. To prevent the system from "collapsing" during these fragments (indeed, the weak safety property does not forbid processes to reset their neighbors lists after each connection/deconnection), we need to specify a stronger safety property guaranteeing that for all the processes whose neighborhood has not changed information is maintained. Specifically, it ensures that there exists a non-empty group of processes for which local information has been maintained between the end of a static fragment and the beginning of the subsequent one. We can see this group of processes as the kernel of the system. More precisely, given two successive configurations c_i and c_{i+1}, with their associated graphs G_i and G_{i+1}, the static common core of G_i and G_{i+1} is the sub-graph common to G_i and G_{i+1} minus all the nodes for which the neighborhood has changed. Formally, let $G1$ and $G2$ be two graphs, and $\Gamma_{G_i}(a)$ the set of neighbors of a in G_i. We define the static common core of $(G1, G2)$ as:

Notation 1

$$Ker(G1, G2) = G1 \cap G2 \setminus \{a : \Gamma_{G1}(a) \neq \Gamma_{G2}(a)\}$$

This leads to the strong self-organization property defined as follows:

Definition 4 (Strong Self-Organization). *Let $\mathcal{S}$ be a system and γ be a global evaluation criterion defined on the configurations of $\mathcal{S}$. A system is strongly self-organized for γ if both the liveness (defined here above) and the strong safety properties hold, with the strong safety property defined as follows:*

strong safety:

$$\forall e = (f_0, \ldots, f_i, f_{i+1}, \ldots), let\ G = Ker(end(f_i), begin(f_{i+1})) :$$
$$\gamma(Proj_{|G}(end(f_i))) = \gamma(Proj_{|G}(begin(f_{i+1})))$$

where $Proj_{|G}(c)$ is the sub-configuration of c including only the states of G nodes.

The following theorem gives a sufficient condition to build a weakly self-organizing system:

Theorem 2 (weak self-organization). *Let $\mathcal{S}$ be a system and γ be a criteria. $\mathcal{S}$ is weak self-organizing for γ if for any process p, $\mathcal{S}$ locally self-organizes in p's neighborhood.*

For space limitations, proof of the theorem is given in [1].

The concept of self-organization can be easily extended to a finite set of criteria. In the following we show that when criteria are not interfering, i.e., when they are independent, then one can build a self-organizing system for a complex criterion by using simple criteria as building blocks. Using the previous example, where the local evaluation criterion was proximity, a second global evaluation criterion is needed to decrease the number of hops of a lookup application, for example we may want to use a few long links to reduce the lookup length.

Theorem 3. *Let $\mathcal{S}$ be a system and let $\gamma_1 \ldots \gamma_m$ be a set of independent evaluation criteria. If S is self-organizing for each γ_i, $i \in [1..m]$ then S is self-organizing for $\gamma_1 \times \ldots \times \gamma_m$.*

4 Case Study P2P systems

We propose peer-to-peer systems as a case study. In these systems, each process (or peer) has the capability to communicate with any other one whatever its geographical position, emphasizing even more the principle of locality. We briefly present a p2p model (see Section 4.1), and show how to build inexpensive p2p applications by locally self-organizing these systems (see Section 4.2).

4.1 Modeling P2P systems

A peer-to-peer system is a collection of peers communicating with each other, and a set of shared resources/data. Peers are physically organized in a weakly connected graph G (i.e., the active peers and links in the system). In order to connect to the system, a peer p executes an underlying connection algorithm which provides p with a set of neighbors (i.e., the peers aware of p). Clearly, graph G may differ at two consecutive instants t and $t+1$ because of the mobility of the system.

Data model. In a pure peer-to-peer network, there is no central server for dispatching information, thus designing a mechanism for peers to communicate is a critical aspect. Data resides locally and the task of efficiently distributing data is non trivial. In previous approaches, such as in Freenet or Chord [8,17], the underlying system duplicates or moves data from one peer to another. In Freenet, complex algorithms are used to achieve anonymity, and in Chord, data are pushed according to their signature, in order to locate them efficiently.

Our approach is different. Instead of moving data along the network, our aim is to dynamically reconfigure the topology of the communication network according to peers similarity. Thus, two similar peers will eventually be connected, and if we assume that the locality principle holds for such systems, we expect a

constant time search. Identifying peers as similar peers relies on the use of similarity functions [13]. Such functions, used locally by a peer to evaluate other peers, are specified according to the system model of data. For example, suppose that the data within the system is strongly structured, then the data similarity function (see Example 1) could be a candidate. On the contrary, if codes are associated to data, then the coding similarity function (see Example 2) could be used.

Example 1 [data similarity] *Let $\mathcal{S}$ be the system, $\mathcal{T}_s$ be the fixed meta-data tree imposed by $\mathcal{S}$, and $\mathcal{T}_{p_1}$ and $\mathcal{T}_{p_2}$ be the trees characterizing the data sets of p_1 and p_2 respectively. The similarity between p_1 and p_2 is the reduced intersection size of $\mathcal{T}_{p_1}$ and $\mathcal{T}_{p_2}$, i.e.: $S_{\mathcal{T}}(p_1, p_2) = \frac{|\mathcal{T}_{p_1} \bigcap \mathcal{T}_{p_2}|}{|\mathcal{T}_{\mathcal{S}}|}$*

Example 2 [coding similarity] *Let $\mathcal{S}$ be the system and h be the coding function defined on $\{K \bigcup \{-\}\}^m$ where K is the keys set and $-$ is the undefined key. Let D_1 and D_2 be two sets of data and v_1 and v_2 be the vector keys associated to D_1 and D_2 respectively ($h(D_1) = v_1$ and $h(D_2) = v_2$). The similarity between D_1 and D_2 is:*

$$s(D_1, D_2) = \begin{cases} 1 - \frac{\sum_{i=1}^{m} |v_1[i] - v_2[i]|}{\sum_{i=1}^{m} v_1[i] + v_2[i]} & \forall v_j[i] \neq -, j = 1, 2 \\ 0, & \forall i, v_j[i] = -, j = 1 \text{ or } 2. \end{cases}$$

P2P system model. We model a P2P system by a dynamic I/O automata. The state of a peer at time t is the value of its local and shared variables and the state of its data. The configuration of the system at t is defined by the state of each peer in the system and the communication graph at t. The transitions of the system are determined by changes in the communication graph, by the invocation of I/O primitives or by the execution of some internal action. Note that an internal action may span the local variables or the owned data.

4.2 Locating Data in a Peer-to-Peer System

Foster [9] emphasis that in many network-based applications, topology determines performance. In order to build an efficient data location service we dynamically organize the topology of the system to gather together peers with a high degree of similar interests. By finding "good" set of peers, we expect to reduce the number of messages broadcast in the network and the number of peers that process a request before a result is found.

Multi-criteria Self-organization We build the data localization service on top of a multi-criteria self-organized layer. This layer is made up of three sub-layers, each of them organizing the topology of the system according to a given criteria. Note that by this layered organization, the retrieval of a data having the maximal similarity with the data owned by a given peer needs $O(1)$ computation steps.

The role of the three sub-layers is now briefly presented. The first sub-layer, the *similarity layer*, guarantees that peers sharing similar data with some particular peer are placed in the neighborhood of this peer. The design of this layer is based on the local evaluation criteria $\mathcal{D}_p$. This criteria maps a peer to a $[0, 1]$ value; intuitively, $\mathcal{D}_p(p') = 0$ when p' and p do not share any data, and $\mathcal{D}_p(p') = 1$ if p and p' own the same set of data. Each peer p maintains a list of peers p' that exhibit the maximal similarity with p, that is, for which $\mathcal{D}_p(p')$ is maximal. Such peers are called *similar neighbors*, and the list containing similar neighbors is denoted $\mathcal{N}^{\mathcal{D}}(p)$. To update this list, p runs the LSA algorithm (see Module 1), with $\mathcal{D}_p$ and $\mathcal{N}^{\mathcal{D}}(p)$ as input. The output is the updated list.

The second one, the *forwarding layer*, gathers together peers exhibiting extreme dissimilarities so that data requests for which a given peer shows no interest are quickly forwarded. The design of this layer relies on the $\mathcal{F}_p$ criteria, which is simply the opposite of the $\mathcal{D}_p$ ones. Analogously to the similarity layer, the $\mathcal{N}^{\mathcal{F}}(p)$ list is built by running the LSA algorithm.

Finally, the *load balancing layer* prevents the islanding phenomenon (or partitioning of the network) from occurring. The load balancing criteria measures the network load. Ideally, the number of neighbors of a peer should be directly proportional to its available capacity or should be in a desired range of it. Let $\mathcal{LB}_p$ be the [0,1]-valued function measuring the load balancing evaluation criteria. $\mathcal{LB}_p(p') = 0$ when p' has no more available capacity. $\mathcal{LB}_p(p') = 1$ if p' is the p's neighbor having the maximal available capacity. As previously, the list $\mathcal{N}^{\mathcal{LB}}(p)$ is built and updated by running the LSA algorithm. Note that since the load balancing criteria maintains the network connectivity, each peer should be connected to this layer.

Theorem 4 (Multicriteria self-organization). *The multi-criteria layer is self-organizing for the composed criteria* $\mathcal{D} \times \mathcal{F} \times \mathcal{LB}$.

Data Localization Layer The data localization layer is built on top of the multi-criteria layer. This layer is responsible for finding needed data and forwarding data request to its neighbors. Module 2 implements these functionalities. Inputs of this algorithm are the $\mathcal{N}^{\mathcal{D}}(p)$, $\mathcal{N}^{\mathcal{F}}(p)$, and $\mathcal{N}^{\mathcal{LB}}(p)$ — neighbors lists for data similarity, data dissimilarity and load balancing criteria respectively.

Three types of messages exist: the data request message, *DR[ID, data, satisfaction_factor]*, the compute similarity message, *CS[ID,data]*, and the evaluation result message, *SR[result]*, where *ID* is the sender's identifier, *data* is the characterization (using trees or codes) of the data the sender is looking for, *satisfaction factor* is the degree of satisfaction settled by the sender, and *result* is the result of the evaluation. The idea of the data tracking algorithm is very simple and is as follows: Upon receipt of a compute similarity message, a peer computes the similarity between its own data and the data in the body of the message, and sends back to the sender the result of the evaluation *SR[result]* (see rule $\mathcal{R}_1$). Upon receipt of a data request message m, peer p computes the similarity between its own data and the data in m. Then p sends to its neighbors a *CS[ID,data]* message for m data. p waits until it collects each evaluation result from its neighbors. If for the requested data, p detects that it has at least one neighbor q which is more similar than itself, then p forwards the *DR[ID, data, satisfaction_factor]* message to q (see rule $\mathcal{R}_2$). On the contrary, if none of p's neighbors are more similar than p and if p's similarity is superior to the *satisfaction factor* then p sends the result of its evaluation to the requester peer. Otherwise, p forwards the request to a neighbor chosen in $\mathcal{N}^{\mathcal{LB}}(p)$).

Module 2 Data Localization Algorithm executed by p (DLA)

Inputs :

$\mathcal{N}^{\mathcal{D}}(p)$: p's neighbors for the criteria $\mathcal{D}$ (data similarity);
$\mathcal{N}^{\mathcal{F}}(p)$: p's neighbors for the criteria $\mathcal{F}$ (data dissimilarity);
$\mathcal{N}^{\mathcal{LB}}(p)$: p's neighbors for the criteria $\mathcal{LB}$ (load balancing);
$\mathcal{D}_p$: the similarity function;

Actions :

$\mathcal{R}_1$: upon the reception of **CS**[ID, data]
send to the neighbor ID **SR**[$\mathcal{D}_p$(data)]

$\mathcal{R}_2$: upon the reception of **DR**[ID, data, satisfaction_factor]
for each $q \in \mathcal{N}^{\mathcal{D}}(p) \bigcup \mathcal{N}^{\mathcal{F}}(p) \bigcup \mathcal{N}^{\mathcal{LB}}(p)$ send **CS**[$\mathcal{D}_p$(data)]
for each $q \in \mathcal{N}^{\mathcal{D}}(p) \bigcup \mathcal{N}^{\mathcal{F}}(p) \bigcup \mathcal{N}^{\mathcal{LB}}(p)$ wait **SR**[result_q]
$result_{max_q} = max_{\forall q \in \mathcal{N}^{\mathcal{D}}(p) \bigcup \mathcal{N}^{\mathcal{F}}(p) \bigcup \mathcal{N}^{\mathcal{LB}}(p)}(result_q)$
if $\mathcal{D}_p(data) < result_{max_q}$ then send to q, **DR**[ID, data, satisfaction_factor]
if $\forall q \in \mathcal{N}^{\mathcal{D}}(p) \bigcup \mathcal{N}^{\mathcal{F}}(p) \bigcup \mathcal{N}^{\mathcal{LB}}(p)$, $\mathcal{D}_p(data) < result\_q$
if $\mathcal{D}_p(data) > satisfaction\_factor$ then send to ID, **SR**[$\mathcal{D}_p$(data)]
if $\mathcal{D}_p(data) < satisfaction\_factor$ then
choose $q \in \mathcal{N}^{\mathcal{LB}}(p)$
send to q **DR**[ID, data, satisfaction_factor]

5 Conclusion & Open Problems

In this paper we have proposed a formal definition of the self-organization property — key feature of the algorithms designed lately for the scalable and dynamic networks. We have exhibited a sufficient condition proving the self-organization of a system. This condition is relying solely on the local knowledge provided by the neighborhood of a process. We have provided a generic algorithm based on the greedy technique which ensures the self-organization of a system with respect to a given input criteria. In addition, we have proposed a multi-layer architectural model for p2p networks. For the sake of illustration we have designed

an intelligent data tracking service on top of this architecture. Clearly, our architecture copes with other services, such as information diffusion, content-based publish/subscribe, each of these services using only a subset of the self-organization layers.

Several problems are open for future investigation. First, the design of the probabilistic extension of our model. This study is motivated by the fact that the connection/disconnection actions are well modeled by probabilistic laws. Essentially, the liveness property should be redefined using the Markov chains model for the probabilistic dynamic I/O automata. Moreover, since we are using a greedy deterministic algorithm, we may only attain a local maximum for the global criterion. Adding randomized choices could be a way to converge (with high probability) to a global maximum. Secondly, we want to design new services such as group information diffusion, content-based publish/subscribe, each of these services using only a subset of the self-organization layers.

Acknowledgement

We would like to thank the referees for all their valuable comments.

References

1. E. Anceaume and M. Gradinariu andM. Roy. Self-organizing systems, case study: peer-to-peer networks. Technical Report PI 1535, IRISA, 2003.
2. J. Aspnes and G. Shah. Skip graphs. In *Proceedings of the 14th annual ACM-SIAM symposium on Discrete algorithms (SODA'03)*, January 2003.
3. P. Attie and N. Lynch. Dynamic input/output automata: a formal model for dynamic systems. In *Proc. of the 20st Annual ACM Symposium on Principles of Distributed Computing (PODC'01)*, pages 314–316, July 2001.
4. H. Attiya and J. Welch. *Distributed Computing : Fundamentals, Simulations and Advanced Topics.* 1999.
5. S Capkun, L. Buttyan, and J. P. Hubaux. Self-organized public-key management for mobile ad-hoc networks. *Transactions on Mobile Computing*, January-March 2003.
6. S. Dolev. *Self-Stabilization.* The MIT Press, 2000.
7. P. Druschel and A. Rowstron. Past: A large-scale, persistent peer-to-peer storage utility. In *HotOS VIII*, May 2001.
8. Freenet. Freenet website. http://freenet.sourceforge.net.
9. A. Iamnitchi, M. Ripeanu, and I. Foster. Locating data in (small-world) peer-to-peer scientif collaborations. *IPTPS 2002*, pages 232–241, 2002.
10. G. Kan. *Harnessing the benefits of a disruptive technology.* O'Reilley & Associates, March 2001.
11. J. Kubiatowicz, D. Bindel, Y. Chen, S. Czerwinski, P. Eaton, D. Geels, R. Gummadi, R. Rhea, H. Weatherspoon, W. Weimer, C. Wells, and B. Zhao. Oceanstore: An architecture for global-scale persistent storage. In *Proc. of the 9th International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS 2000)*, November 2000.
12. Jonathan Ledlie, Jacob Taylor, Laura Serban, and Margo Seltzer. Self-organization in peer-to-peer systems. In *10th European SIGOPS Workshop*, September 2002.
13. M. Li, X. Chen, X. Li, B. Ma, and P. Vitanyi. The similarity metric. *Proceedings of the 14th annual ACM-SIAM symposium on Discrete algorithms (SODA'03)*, pages 863–872, 2003.
14. S. Ratnasamy, P. Francis, M. Handley, and R. Karp. A scalable content-addressable network. *Proc. of the ACM SIG/COMM*, pages 161–172, 2001.
15. A. Rowstron and P. Druschel. Pastry: Scalable, distributed object location and routing for large scale peer-to-peer systems. *Proc. of the 18th IFIP/ACM International Conference on Distributed Systems Platforms (Middleware 2001)*, 2001.
16. A. Rowstron and P. Druschel. Storage management and caching in past, a large-scale, persistent peer-to-peer storage utility. In *Proc. of the 17th ACM Symposium on Operating Systems Principles (SOSP)*, pages 188–201, 2001.
17. I. Stoica, R. Morris, D. Karger, M.F. Kaashoek, and H. Balakrishnan. Chord: A scalable peer-to-peer lookup service for internet applications. In *Proc. of the ACM SIG/COMM*, pages 149–160, August 2001.
18. J.E. Walter, J. L. Welch, and N. M. Amato. Distributed reconfiguration of metamorphic robot chains. *Proc. of the 19th Annual ACM Symposium on Principles of Distributed Computing (PODC'00)*, pages 171–180, 2000.
19. H. Zhang and A. Arora. Gs3 : Scalable self-configuration and self-healing in wireless networks. *Proc. of the 21st Annual ACM Symposium on Principles of Distributed Computing (PODC'02)*, pages 58–67, 2002.
20. B. Zhao, J. Kubiatowicz, and A. Joseph. Tapestry: An infrastructure for fault-tolerant wide-area location and routing. Technical Report UCB/CSD-01-1141, Computer Science Division, U. C. Berkeley, April 2001.