SONDe, a Self-Organizing Object Deployment Algorithm in Large-Scale Dynamic Systems

Vincent Gramoli
EPFL - Université de Neuchâtel
Vincent.Gramoli@epfl.ch

Anne-Marie Kermarrec
INRIA-Rennes Bretagne Atlantique
Anne-Marie.Kermarrec@inria.fr

Erwan Le Merrer
INRIA - Orange Labs
elemerre@irisa.fr

Didier Neveux
Orange Labs
Didier.Neveux@orange-ftgroup.com

Abstract

We present the design, correctness, and analysis of SONDe, a simple fully decentralized object deployment algorithm for highly requested systems. Given an object *(service or data), SONDe provides a node with a constant upper bound (h) on the number of logical hops to access an object holder (*provider*), thus making tunable and predictable the communication latency between a node and any provider. In addition, SONDe is able to dynamically adapt the number of providers to reflect load variations experienced in localized portions of the system. Each node individually decides to be a provider, based on the observation of its h-hops neighborhood. We show theoretically that SONDe self-stabilizes and provides an independent-dominating set of providers. Finally simulation results, conducted over different network topologies, demonstrate the efficiency of the approach and confirm the theoretical analysis.*

1. Introduction

A peer to peer (P2P) overlay network organizes a large set of participants (*nodes*), in a logical network on top of a physical network topology. The scalability of P2P networks relies on the fact that each node may act both as a client and a server. Therefore, the number of potential servers grows linearly with the size of the system. For a given application to provide a service, a subset of nodes may be elected to act as servers, to which other nodes can forward requests. With the adaptation of service access applications in P2P networks, one of the major challenges is to deploy an *object* (service or data) so that it becomes rapidly accessible from any node of the network.

So far, most of P2P applications have been devoted to data access, like file sharing applications. In such applications, as soon as a node acquires a data locally, it becomes a provider for this data to other nodes. Only recently, P2P applications have started to focus on service access, such as for instance Internet telephony applications. In these applications, all nodes have the ability to provide the service at any time and protocols such as SIP [19] suffer from highly accessed centralized services. Conversely, distributed telephony systems [3, 18] require a dedicated service deployment mechanism. Ideally, this deployment should minimize the resource consumption by limiting the number of nodes running the service. In the meantime, this deployment should allow service availability to fulfill quality of service requirements. Here, we precisely address the problem of object deployment where each object may represent either a data or a service.

This paper proposes a solution to the object deployment problem by selecting nodes, also called *providers*, that provide the object to other nodes and guarantee its availability in a self-organized manner. This deployment organizes the density of providers such that any other node accesses the corresponding object within bounds (with respect to number of hops and/or delays). The access rate variations are handled through a dynamic adaptation of the provider density in response to bursts of load.

Contributions. The proposed solution is a dynamic and fully decentralized object deployment protocol, called SONDe (*Self-Organizing Network Density*). SONDe ensures object availability while minimizing the number of object providers. More precisely, SONDe ensures that an object is at a tunable constant number of hops from any node in the system. SONDe is fully decentralized and highly scalable as nodes implement the object deployment algorithm based only on the observation of their h-neighborhood.

Even though the distribution of objects maps the require-

ments (all nodes have a provider in h hops, and any pair of providers cannot see each other), it might happen that some providers are overloaded as the usage suddenly increases in a given neighborhood. To overcome this, each provider in SONDe may take specific actions to adapt to such situations by adapting the search depth in the overloaded area. Therefore, SONDe, in a network-independent fashion, balances the load automatically between the system nodes. If a burst of load occurs in a localized area of the overlay, the number of providers is automatically increased in this specific area. Conversely, when the load decreases the number of providers is reduced accordingly in order to limit the unnecessary resource consumption.

The paper shows by extensive experimental analysis that the resulting algorithm adapts rapidly its provider density to support load variation and tolerates high dynamism. Moreover, SONDe is proved to self-stabilize to an h-independent-dominating set of providers. An h-independent-dominating set of providers is a set of providers such that no two providers are closer than distance $h + 1$ from each other, and every node has at least a provider in at most h hops. The notion of independence limits the number of providers while dominance allows predictable access time[1] from any node to a provider.

Background. Distributed Hash Table (DHT) [16, 20, 17] is a well-known solution to distribute and locate an object in a P2P system. In DHT-based solutions, nodes and object get identifier in a global identifier space. Objects are automatically assigned to providers in this space. For example, in Pastry [17], the provider of an object is the node whose identifier is the numerically closest to the object identifier. Such a systematic assignement have some load balancing isues, if a few objects are accessed by many nodes simultaneously, heavy congestion/overload might occur at the targeted providers. Second, a node might be located far from the targeted provider, and thus accessing the object might be costly. Hence, DHT-based solutions are not suitable for the tunable object deployment problem.

Sortie [8] provides a self-organizing object deployment mechanism based on the demand; the objects are deployed close to the request locations. Unfortunately, Sortie does not take into account maximal hop distance between providers and other nodes.

On the theoretical side, obtaining small dominating set in a graph remains an important issue. The problem of finding the minimum dominating set has been shown to be NP-hard [5] and an easier problem is to find a small (but not minimum) dominating set. In [9], the authors propose an algorithm that constructs small dominating set in a logarithmic number of cycles, while in [10], the authors construct a small dominating set in constant time. The same problem has already been investigated in the context of wireless sensor networks [7, 14] where the diameter of the graph increases as the considered area is scaling. P2P allows various forms of communication graphs that generally present low diameter despite their size. Importance of dominating set for P2P systems have been outlined by some specific applications [22]: the authors aim at maximizing the number of responses to an object search in a P2P system. This approach does not focus on providing the object at a constant number of hops. Finally, in [2] the authors focus on both independent and dominating sets, and propose a centralized solution to this problem. In the following, we generalize the placement problem to neighborhood up to distance $h \geq 1$ and solve it in a distributed manner.

Roadmap. Section 2 presents the model and the problem addressed in this paper. SONDe algorithm is described in Section 3, proved correct in Section 4, and evaluated in Section 5. Finally Section 6 concludes the paper.

2. Design rationale

2.1. System model

The system consists of a connected set of n nodes. Each node is uniquely identified and does not maintain any global information about the system. Nodes can join and leave the system at any time. The P2P overlay network is represented as a communication graph, denoted $\mathcal{G}$, in which a node i can communicate directly with a subset of nodes called its direct neighbors. More generally, we refer to the *neighbors* of i, as the set of nodes $\mathcal{N}_i^h$, located at the furthest at distance h from i in the logical network, where $h \in \mathbb{N}^+$ is a constant. In other words, the minimal distance between i and any $j \in \mathcal{N}_i^h$ is lower than or equal to h. $\mathcal{G}$ is an undirected graph such that communication links are symmetric, formally if $i \in \mathcal{N}_j^h$ then $j \in \mathcal{N}_i^h$. The number of neighborhoods that a node belongs to is thus directly related to h and to the *density* (namely the quantity of edges) of the communication graph.

A service or a data used in the system is equally referred to as an *object*. In the remaining, we only focus on a single replicated object. The communication delay between two neighbors is denoted δ.

We assume that every node is able to act as an object provider. Therefore becoming a service provider is achieved by simply switching on this functionality; how to actually acquire an object is out of the scope of this paper. We refer to a node that holds an object as a *provider*, other nodes are referred to as *clients*. Providers directly access to their own object locally.

[1] For accessing purpose, an underlying logical/physical layer mapping mechanism could be used jointly to our algorithm, see e.g. [13].

2.2. Problem Statement

The algorithm presented in this paper addresses the issue of limiting the number of providers of a single object x while providing any node with an object replica located nearby (in the h-vicinity). First, provided the object x consumes computation resources on the providers, limiting the replication of x over the network saves resources for other objects. Second, providing any node with an object x at a small distance helps finding and accessing x quickly. To formalize those two challenges we borrow two well-known definitions from graph theory: *independence* and *dominance*.

The independence notion expresses the idea of restricting the number of providers. Consider an undirected connected graph $G = \langle V, E \rangle$. A subset of vertices $\mathcal{I} \subseteq V$ of the graph G is called an *independent set* if and only if there is no edge in E between two vertices of $\mathcal{I}$. In the communication graph $\mathcal{G}$, if the set of providers of x forms an independent set, then x does not consume computation resources of all nodes.

The dominance notion expresses availability. Consider an undirected connected graph $G = \langle V, E \rangle$. A subset of vertices $\mathcal{D} \subseteq V$ of the graph G is called a *dominating set* if and only if it exists an edge between any vertex of $V \setminus \mathcal{D}$ and a vertex of $\mathcal{D}$. Observe that a solution to this problem is $\mathcal{D} = V$. Like above, consider the communication graph $\mathcal{G}$. If the set of providers of x is dominant in $\mathcal{G}$, then x is easily available from any location.

Traditionally, those definitions apply to vertices whose distance is 1. Depending on the application requirements, this might be too restrictive to apply this on nodes that are located a single hop away from each other. Here, we relax such a constraint by letting the programmer define an edge between two vertices (nodes) if their distance is smaller than h hops.

Let an *independent-dominating set* be a set $\mathcal{S}$ satisfying both definitions. As said before, we extend the aforementioned definition to the h-neighborhood. Let E^h be the set of path containing at most h consecutive edges of E. Definitions of h-*independent* and h-*dominating* sets rely on the same definitions, with augmented set of edges E^h.

Definition 2.1 (h-Independent-Dominating set) *Let $G = \langle V, E \rangle$ be an undirected connected graph. A subset of vertices $\mathcal{S} \subset V$ is an* h-independent-dominating set *iff*

$$\begin{cases} \forall (u,v) \in E^h \Rightarrow u \notin \mathcal{S} \vee v \notin \mathcal{S}, \\ \forall u \in V \setminus \mathcal{S}, \exists v \in \mathcal{S} \text{ s.t. } (u,v) \in E^h. \end{cases}$$

The problem of independent-dominating set is also known as the maximal independent set. While an ideal solution would be a minimum dominating set that is NP-hard to compute, our solution provides availability with a relatively low complexity.

3. SONDe algorithm

This section describes the SONDe algorithm. Subsection 3.1 illustrates how SONDe ensures that a node can contact a provider in its neighborhood $\mathcal{N}_i^h$ without global knowledge. We then add a heuristic for load tolerance in Subsection 3.2.

3.1. Toward a h-indepedent-dominating set

To converge to the property where every node can contact a provider in $\mathcal{N}_i^h$, each node verifies whether a provider is located the furthest at h hops. This *verification* is done periodically (unsynchronized), with period $\rho \geq \delta$, through message exchange. To limit communication overhead, only providers respond.

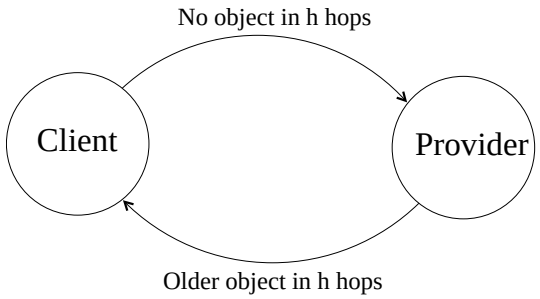

Figure 1. SONDe states.

After each verification, the corresponding node updates its state as indicated on Figure 1. If i is not yet a provider and no provider is found in $\mathcal{N}_i^h$, then i becomes a provider (cf. Line 22 in Algorithm 1). Conversely, if the requester i is a provider and another provider is found in $\mathcal{N}_i^h$, then i either stop providing or keep providing, depending on some specific policy described below.

When two nodes in the same h-neighborhood have not found any provider by verifying simultaneously, they naturally both decide to become provider. To speedup the convergence to a provider state or to a client state, each node records the time at which it became last a provider. This recorded time coupled with the node ID, say the *age* of the provider, is used as a symetry breaker to ensure the convergence of the algorithm. Every provider piggybacks its age in its response message and the oldest of the two providers remains provider while the youngest switches its state back to client. In other words, the provider stops providing and becomes a client only if it finds an older provider than itself (cf. Lines 28–29).

3.2. Load tolerance

Although the solution, after convergence, ensures that a provider is present in each vicinity, some providers may

Algorithm 1 SONDe algorithm at node i

Prerequisite Functions:

setHLocal$(k)_i$ notifies a constant set of neighbors at h_{local} and sets the communication range of node i, h_{local}, to k.

isOvrld$()_i$, isUndld$()_i$, and provider$()_i$ return true if the node i is overloaded, underloaded, or its status is $provider$, respectively, false otherwise.

findProvider$(h_{local})_i$ returns true if there is a *provider* in $\mathcal{N}_i^{h_{local}}$, false otherwise.

findOlderProvider$(h_{local})_i$ returns true if there is an older *provider* with the same h_{local} value in $\mathcal{N}_i^{h_{local}}$, false otherwise.

becomeClient$()_i$ sets the status of node i to $client$.

becomeProvider$()_i$ sets the status of node i to $provider$.

getNbrAvg$()_i$ returns the average of the h_{local} values of the peer's neighbors.

provider$()_i$ returns whether i is a *provider*.

Pseudocode of the Algorithm at node i:

```
20:  if ¬provider() then
21:    if ¬findProvider(h_local) then
22:      becomeProvider()
23:    else
24:      unchgCycles ← unchgCycles + 1
25:    if unchgCycles > unchgThreshold then
26:      h_local ← getNbrAvg()
27:  else   // Current node i is a provider.
28:    if findOlderProvider(h_local) then
29:      becomeClient()
30:    else
31:      if isOvrld() then
32:        ovrldCycles ← ovrldCycles + 1
33:        undldCycles ← 0
34:      else if isUndld() then
35:        undldCycles ← undldCycles + 1
36:        ovrldCycles ← 0
37:      else   // normal state
38:        ovrldCycles ← 0
39:        undldCycles ← 0
40:      if ovrldCycles > ρ_over then
41:        setHLocal(h_local − 1)
42:        ovrldCycles ← 0
43:      else if undldCycles > ρ_under then
44:        setHLocal(h_local + 1)
45:        undldCycles ← 0
```

suffer from bursts of load. Indeed, providers become overloaded if there is a large number of clients in their vicinity and if these clients are using too heavily its resources. Such load pattern may affect the quality of service of the application by delaying responses or even losing requests.

To cope with this load issue, SONDe uses a mechanism to increase the number of providers in an overloaded network area, keeping in mind simplicity and local processing only. consequently, in addition to the h parameter already defined, we introduce a second parameter, h_{local}, with each node. The h_{local} parameter is such that $h_{local} \leq h$ and represents the node current verification depth bound. This parameter is periodically updated regarding to the load variations. When the parameter h_{local} of node i decreases (resp. increases), then the verification range of node i decreases (resp. increases) accordingly.

Next, the load level applied to provider i is recorded using two additional fields: Θ_{over} and Θ_{under} ($0 < \Theta_{under} < \Theta_{over} < 100$) indicate the load threshold over which and under which, respectively, the load level is no more acceptable for provider i (this percentage is returned by the provider's operating system, and is used by the isOvrld()/isUndld() fonctions in Algorithm 1). We assume here that all nodes have identical thresholds for simplicity in the presentation, however, we claim that the heterogeneity of thresholds would not affect our algorithm execution.

In case the load level of a provider i remains over Θ_{over} during a predefined period ρ_{over} of time, the provider decreases its load by reducing the h_{local} parameter of its neighbors. More precisely, provider i asks a fixed number of direct neighbors (typically $\log n$) to set their h_{local} to the provider current one decremented by one; so does the provider itself (cf. Line 41). The consequence is that at the next verification, some of these neighbors create new providers as they are not able to find other providers at this reduced range. That is, this action increases the density of providers in the overloaded area.

In contrast, if a provider i remains underloaded (Θ_{under}), during a period of time at least equal to ρ_{under}, then it notifies its $\mathcal{N}_i^{h_{local}}$ (as in the overloaded case) to update their h_{local} to the value of its own incremented by one, and updates its own. This process naturally decreases the

density of providers in the underloaded area.

Remarks. The aim of ρ_{under} and ρ_{over} values is to create a *hysteresis* phenomena and to stabilize the provider density by limiting the number of state changes. This diminishes the sensitivity of the network to slight load variation and priviledges sentivity to only significant load variation.

During the verification process, a node stops providing only if it finds in $\mathcal{N}_i^{h_{local}}$ another provider with the same h_{local} parameter; we thus relax here the independance constraint for providers with different h_{local}. This preserves the previous h_{local} changes. Contrariwise, a node becomes provider only if it finds no other provider in h_{local} hops. This mechanism ensures that an isolated load variation does not impact on the whole overlay, but only on a local area of the overlay.

Finally, nodes that have not been asked to increase their h_{local} during a load decrease simply regularly set their h_{local} field to the average of their direct neighbor values, to adapt their search bound to the local representative one.

4. Proof of self-stabilization

The purpose of this section is to show that the SONDe algorithm self-stabilizes in that the set of providers self-stabilizes to an h-independent-dominating set. We do not claim that the convergence time is optimal, we simply shows that the algorithm self-stabilizes. In different contexts, there are probabilistic self-stabilizations with a convergence time depending on the maximal degree of the graph [7]. We do think that SONDe is practical on peer-to-peer communication graphs where the degree is generally high and the diameter is generally low.

Here are some preliminary notations for the proof. Let $\mathcal{L}$ be the set of all legitimate configurations where the communication graph $\mathcal{G}$ admits a set $\mathcal{S}$ of providers such that $\mathcal{S}$ forms an h-independent-dominating set since 2δ time. By abuse of notation, we say that a graph is h-independent-dominating if and only if the set of its providers represent an h-independent-dominating set. Assume that the system stabilizes such that dynamics stop, in-transit messages represent the state of their senders, messages are exchanged every $\rho = \delta$ time, a message exchange lasts exactly δ time, and nodes are neither overloaded nor underloaded. Consequently, for any node, h_{local} does not change over time and remains equal to h. We show that starting from any possible configuration, the algorithm converges to a configuration of $\mathcal{L}$ (convergence) and that when such a configuration is reached the system stays in a configuration of $\mathcal{L}$ (closure).

For the sake of the convergence proof, we observe the system at separate intervals of time 2δ and we refer to the distance D from a legitimate configuration as the sum of: *(i) non-dominating factor*, (d_1), the number of clients that do not have a provider in their h-neighborhood. *(ii) dependence factor*, (d_2), the number of links that are between two providers plus the number of links adjacent to any of these links. As we said, the system is observed at separate intervals of 2δ time, that is, a providers must be a provider during at least two subsequent observations. Observe that neither d_1 nor d_2 can be negative. That is, if $D = 0$ (i.e., $d_1 = d_2 = 0$) since 2δ time, then the set of providers forms an h-independent-dominating set.

Figure 2 represents the self-stabilization of SONDe to a worst case scenario (with maximal provider density) where the distance $D = d_1 + d_2$ keeps decreasing. Such a worst-case is rare in practice for two reasons: *(i)* the communication graph is generally not a star (this clearly does not scale for large dynamic overlays), and if so, *(ii)* the center of the star might be the older provider leading to a minimal provider density.

Lemma 4.1 (Convergence) *While the distance D to a legitimate configuration is not null, D decreases at each interval of 2δ time.*

Proof. Without loss of generality, we consider the system at time t and $t' = t + 2\delta$. Let d_1 (resp. d'_1) be the non-dominating factor at times t (resp. t') and d_2 (resp. d'_2) be the dependence factor at times t (resp. t'). Since messages are not lost, if $d_2 \neq 0$ then providers that are neighbors at time t have successfully exchanged a message by time t'. Let p be the number of providers that change their state by time t'. Since all p providers have distinct ages, at least one provider remains a provider at time t' (the oldest). By definition of d_1 and d_2, if this increases the non-dominating factor by $p-1$, then this decreases the dependence factor by at least p. (Independently, this would result in $d'_1 = d_1+p-1$ and $d'_2 \leq d_2-p$.) Differently, if $d_1 \neq 0$ then c clients with no provider in their neighborhood may decide to become provider. This increases the dependence factor by less than or exactly c. (If this happens meaning that $d'_1 = d_1 - c$, then $d'_2 \leq d_2 + c$.) Consequently, these two processes lead to $d'_1 + d'_2 \leq d_1 + d_2 - 1$, and the result follows. □

Invariant 4.2 (Closure) *If the system reaches a legitimate configuration then it remains in a legitimate configuration.*

Proof. By definition of a legitimate configuration, the set of provider forms an h-independent-dominating set since 2δ time. That is, the information each provider receives is up-to-date. Since dominance ensures that each client has a provider in its neighborhood, and independence ensures that no providers are neighbors the result follows. □

Theorem 4.3 *SONDe algorithm self-stabilizes.*

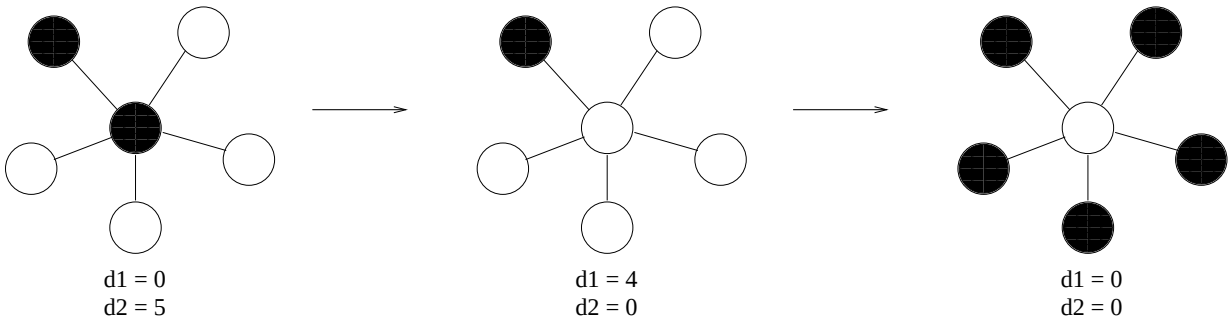

Figure 2. An example of a SONDe execution leading to a maximal provider density. 2δ time interval between observations, $h = 1$, black disks: providers, white disks: clients.

Proof. The result is straightforward from the convergence property ensured by Lemma 4.1 and the closure property ensured by Invariant 4.2. □

5. Evaluating SONDe

5.1. Provider neighborhood size

In SONDe, the number of clients accessing the same provider is of crucial interest. Effectively, this number indicates the potential request rate and load a provider may experience. Consequently, an interesting question is: for a given degree k and a given hop count h, what is the number of nodes that can access a single provider?

To answer this question, assume that every node i has exactly a constant number k of neighbors, where k could be seen as the graph degree max bound. Consequently, the set of nodes can be seen as a tree rooted in i of depth h and width $k-1$ (except at the root level where width is k). The number r of nodes is equal to $k \times |T_{k-1,h-1}| + 1$ where $T_{k-1,h-1}$ is a tree of depth $h-1$ and width $k-1$. It is easy to show by recurrence that tree $T_{k,h}$ contains $\frac{k^{h+1}-1}{k-1}$ nodes. Consequently, we obtain: $r = \frac{k((k-1)^h-1)}{k-2} + 1$.

SONDe allows to define the maximal load applied to providers, by tuning the parameter h. For example, if every node i has degree 4 and contacts all the nodes at distance at most 4, then each provider covers $\frac{4(3^4-1)}{2}+1 = 161$ nodes.

5.2. Experimental setup

To evaluate SONDe performance, we implemented SONDe using the PeerSim simulator [15], in a cyle based configuration. Following results are based on 10^4 node networks; scalability considerations are specifically addressed in Subsection 5.5. SONDe is evaluated on three different topologies: *(i)* a 2-dimensional network (2-D), *(ii)* a homogeneous random network and *(iii)* a scale-free network. The results outline the good behavior of SONDe on those representative topologies. For the sake of comparison of the three distinct topologies, we force the mean degree to be of $\log n$, i.e. each node has roughly $\log n$ neighbors on average (here 4); the total number of edges in each topology is thus $20,000$. These settings provide a fair comparison on the output results.

2-dimensional network. The first topology is a simple 2-dimensional overlay, where each node is located in a 2-dimensional coordinate space. Every node is connected to 3 to 6 (3.8 in average) nearby nodes. Neighbors are chosen preferably by their closeness in terms of the cartesian distances in the coordinate space. This topology is particularly useful to visually validate the homogeneous repartition of the providers and to observe the impact of local load events on the rest of the overlay in terms of replication reaction. Moreover, mapping the logical overlay to the physical one minimizes in practice the overall communication latencies in a real world implementation.

Homogeneous random network. The second topology is a homogeneous random network, where each node has a constant number of neighbors, chosen uniformly at random. During the construction of the topology, a node chooses a neighbor (selected uniformly at random) if the contacted node does not have already the maximum number of edges. Otherwise, the process keeps going until each node obtains the desired amount of neighbors. The number of neighbors chosen here is 4.

Scale-free network. The last topology a scale-free network as defined by Barabási and Albert [1]. The degree of nodes of a scale-free network follows a power law distribution. Such networks are now widely used to model social

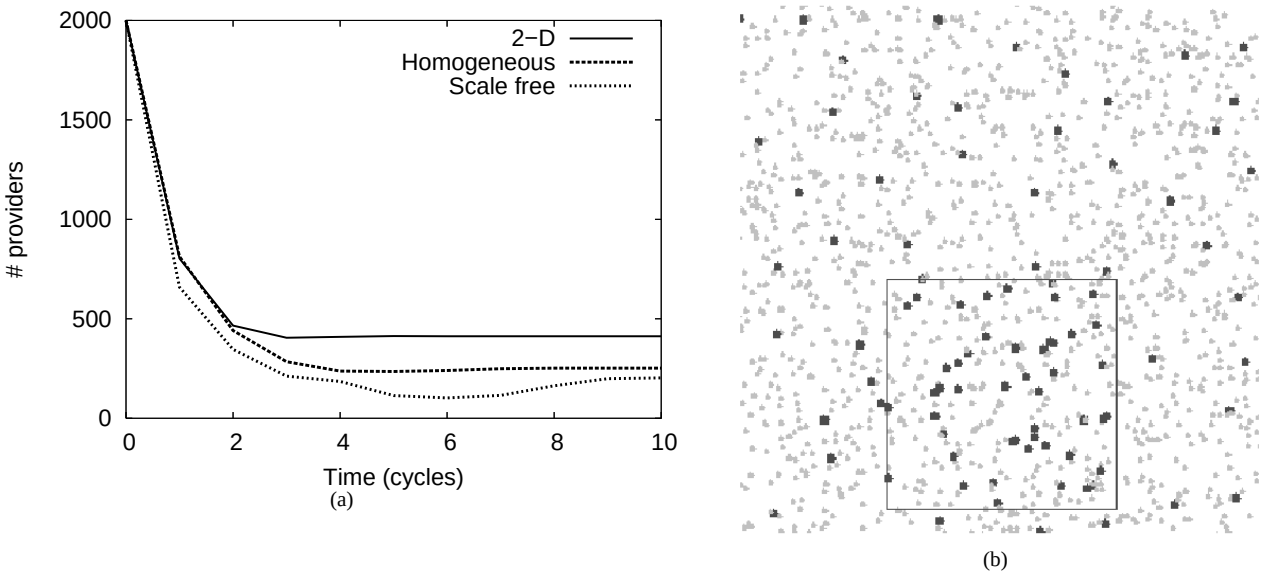

Figure 3. (a) Convergence toward a stable state, 20% of nodes simultaneously verifying $\mathcal{N}_i^h$, 10^4, 2-D unstructured overlay. (b) Distribution of the providers on the 2-D overlay; upper part: no load applied; lower part: overloaded region in the square overlay.

networks such as the Internet, and peer-to-peer networks. Peers have between 2 and 230 neighbors (3.99 in average).

In the remaining of this section, we call a *cycle* the time needed by all nodes of the overlay to accomplish their verification and h set to 4.

5.3. Providing an available object

Here, we show that SONDe stabilizes rapidly to a state where each node is provided with an object nearby, and no two providers are closer than distance $h+1$ from each other, system thus reaching a h-independent-dominating set.

Figure 3(a) indicates the convergence time of SONDe when applied to the three topologies. At the beginning of the execution, no providers are present in the system. Then, nodes concurrently verify at each cycle the presence of providers in their neighborhood: 20% of nodes verify at the same time if the rest of node act unsynchronized. Recall that the set of providers represents an h-independent dominating set in the steady-state. At first sight, this figure confirms the theoretical proof of Section 4: SONDe self-stabilizes. Furthermore, all curves indicate that despite high concurrency, only a few number of cycles are necessary for the nodes to reach a steady-state. Hence, convergence time is fast compared to the system size.

From a closer look, we observe the impact of the topology on the number of providers required. Recall in all three cases, the topologies have the same number of nodes and very similar edge density. The 2-D topology has the largest diameter and the largest number of providers (in the steady-state) whereas the scale-free topology has the lowest diameter overlay and the lowest number of providers (in the steady-state). In the latter case, as the neighborhoods of nodes are likely to be wider (presence of highly connected nodes, called *hubs*), convergence time is slightly increased.

Figure 3(b) visualizes graphically the provider density. For this purpose, this figure represents the providers as large black dots and the clients as small grey dots as they are placed in the 2-D topology. For the sake of clarity, links are not depicted. On the upper side of the picture no specific burst of load occurs. We can see on this part that the providers are homogeneously distributed, so that the density is even on the network; this is one of the nice features of an independent-dominating set.

5.4. Tuning provider density

Figures 4(a) and 4(b) show the system behavior when the load is increased gradually. The number of clients accessing the objects simultaneously is represented on the left-hand side y axis. The x axis represents the time. Figure 4(a) depicts the number of providers created in response to the load variation. The load is gradually increased from cycle 1000 to cycle 5500 where all nodes (i.e., 10^4 nodes) are simultaneously querying a provider. The considered provider capacities are 5, 10 and 25 active clients per provider. The

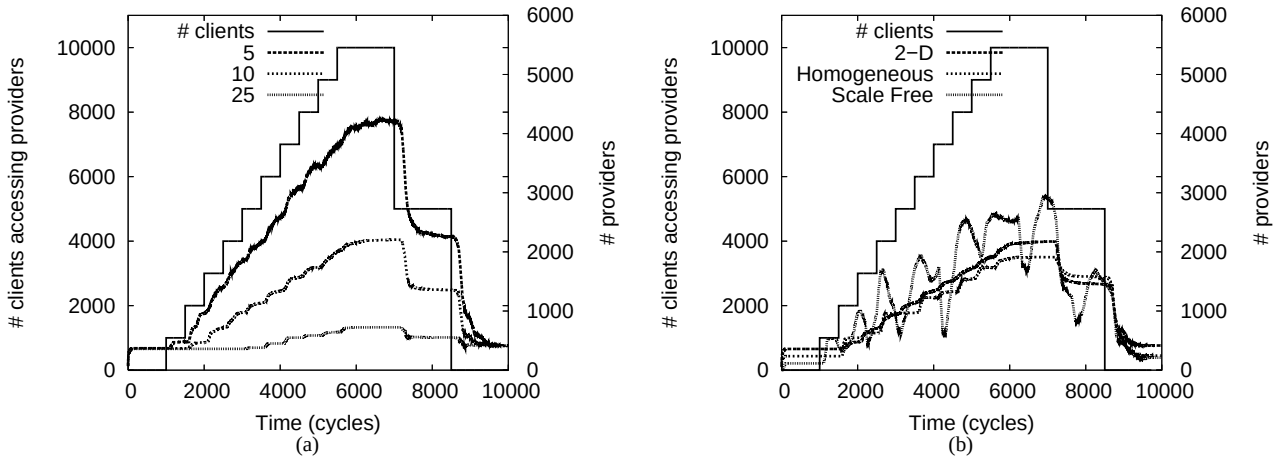

Figure 4. (a) Object creations in reaction to load variations, under different provider capacities, h=4, 10^4 nodes, 2-D unstructured overlay. (b) Evolution of the number of providers in overlays under access rate variation, h=4, 10^4 nodes.

load thresholds, Θ_{under} and Θ_{over}, are set to 30% and 70% of the provider capacity. For instance, if the capacity of the provider is 10, then its underload threshold, Θ_{under}, is 3 while its overload threshold, Θ_{over}, is 7 clients/provider. When the load decreases, the algorithm suppresses underloaded providers to save resources. In consequence, as soon as a burst of load stops, the system goes back to a basic steady-state. As expected, simulations show on Figure 4(a) that an overlay composed of more powerful providers triggers less replica creations when facing load increase.

In the rest of the experiments, we set the number of clients by provider to 10. Figure 4(b) shows the increase of the number of providers under the same load pattern, for the three topologies introduced earlier; Θ_{over} is set to 10%. This value allows a quicker convergence towards a stable state (around t=6500), as this leaves more providers in the system when load decreases, because the load has to be very low to trigger provider deletion for resource saving. We observe that the number of created replicas smoothly adapts to the current load access applied on the system. The scale-free curve is noisier than the two others as the stabilization time is increased; this is due to the fact that the configuration of replicas placement is harder to reach on this topology in presence of an important number of replicas, as the neighborhood of each node is wider.

During these simulations and for the three topologies we observed that the percentage of satisfied nodes (i.e., the nodes that are able to find a non overloaded provider in $\mathcal{N}_i^{h_{local}}$) is close to 100% at any time, despite load increases; the goal of availability of the algorithm is thus reached. The cause for this overall good behavior is twofold: *(i)* generally a node has several providers in $\mathcal{N}_i^{h_{local}}$ (thus have a larger chance to find a non saturated provider), and *(ii)* SONDe reacts rapidly to the variation of load.

An important parameter as far as load increase is concerned is to what extent the density increase remains localized in the overloaded part of the network. To measure this area of increased density, we applied a load increase at some providers located in a subspace of the 2-D coordinate space. Figure 3(b) represents the 2-D topology where the load increase is only applied to some providers of a subspace. This subspace is represented by a square at the bottom of the figure. We can clearly see on this figure that the density of provider is larger inside the square than outside. This is due to the fact that a load event in a part of the overlay is absorbed by its close neighborhood, preventing the load increase from affecting the whole overlay.

5.5. Scalability

Although we presented the results obtained in a system of 10^4 nodes, we scaled up the simulation to 10^5 and 10^6 nodes. The number of providers obtained in the 2-D topology on a non loaded network was respectively 2434 and 24363, while there were 249 providers in a system of 10^4 nodes (see the plotted runs). This shows that the number of providers increases linearly in the size of the system, following the expectation that the use of only limited knowledge of the system on each node gives the key to algorithm scalability. The fact that SONDe ensures local properties such as the presence of an object in each local subgraph of $\mathcal{G}$ entails that the network scale has no impact on the overall

algorithm behavior.

5.6. Handling churn

For the purpose of tolerance to dynamism, we simulated a churn prone environment where a constant number of nodes join and leave the system continuously. Results are depicted on Figure 5.

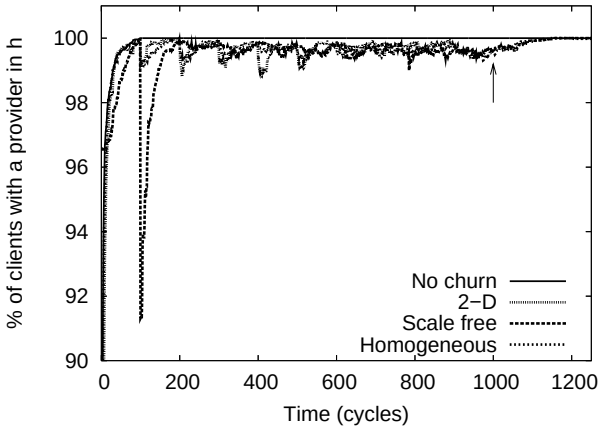

Figure 5. Percentage of satisfied nodes (able to reach a provider within $\mathcal{N}_i^h$) under churn (stopped at t=1000), h=4, 10^4 nodes.

Stutzbach et al. [21] highlighted that session lengths in P2P system are consistent across systems like Gnutella, Kad, BitTorrent: almost all nodes have left after one day. In order to apply a realistic churn in our cycle-based simulations, we take the session length for half of the nodes; after 30 minutes, all the nodes have left the overlay, always replaced by new nodes. If we take a duration of 1 second for a cycle in our simulations, we have thus a churn of 0.055% of the nodes per cycle. When the number of direct neighbors of a node falls under a predefined minimum value, the node looks for another contact node as new neighbor (to keep the communication graph connected).

Figure 5 compares the static case to the dynamic case: the "No churn" curve plots the 2-D topology case without churn, while the three other curves correspond to the three studied topologies with churn. More precisely, this figure shows the impact of such a configuration on the proportion of node having a provider in their h-vicinity in the three considered topologies. The curves present the percentage of clients i that have found a provider within $\mathcal{N}_i^h$, starting from a network provider-free. Only a few cycles are required for the rate of unsatisfied nodes to lie within 1%. Churn is stopped at cycle 1000; we observe that the system converges quickly to a steady-state forming the h-independent-dominating set of providers.

5.7. Neighborhood exploration

In this paper, we did not make any specific assumption on the exploring method used to search a provider in a node's neighborhood during a verification. Several approaches could be considered differing by the traditional trade-off between latency and overhead. Basic search techniques such as flooding in unstructured P2P overlays, are expensive in terms of overhead; techniques like *Expanding Ring* [12] or *Random Breadth-First-Search* [4] with a fixed TTL could for example be used to significantly improve the overall performance.

As a lightweight alternative, providers could notify their h-neighborhood about their presence, carrying a distance information increased at each hop, in order for their neighbors to evaluate if the $\mathcal{N}_i^{h_{local}}$ condition is satisfied. Such a reactive heuristic would avoid each client to periodically perform a neighborhood verification. Nevertheless, this solution would be less reactive than the original one, since each client should wait before being able to access an object.

6. Conclusion

In this paper, we have presented the design, correctness, and analysis of a simple object deployment algorithm, SONDe, for large-scale dynamic systems. SONDe has been designed with simplicity in mind and does not require any specific overlay structure. We presented SONDe using a single replicated object, however, multiple objects (or services) could be provided in parallel. SONDe was also used for its homogeneous providers repartition capacities, to serve as a basis for the construction of a routing backbone in a WSN system [11]. First, SONDe ensures availability of provider at any location in the system. Thus, SONDe minimizes the scope of search required to find a provider. Second, SONDe tunes provider density in a region where the workload varies to prevent providers from being overloaded.

Future work includes SONDe implementation on a large-scale distributed execution plateform, to measure impact of message loss or congestion. The gossip-based communication behavior of SONDe may allow developers to use GossiPeer [6], a toolkit for distributed gossip-based experimentation, to implement it in a simple manner.

References

[1] R. Albert. *Statistical mechanics of complex networks*. PhD thesis, University of Notre Dame, Notre Dame, Indiana

46556, 2001.

[2] P. Alimonti and T. Calamoneri. Improved approximations of independent dominating set in bounded degree graphs. In *WG '96: Proceedings of the 22nd International Workshop on Graph-Theoretic Concepts in Computer Science*, pages 2–16, London, UK, 1997. Springer-Verlag.

[3] D. A. Bryan, B. B. Lowekamp, and C. Jennings. SOSIMPLE: A serverless, standards-based, P2P SIP communication system. In *AAA-IDEA '05: Proceedings of the First International Workshop on Advanced Architectures and Algorithms for Internet Delivery and Applications*, pages 42–49, Washington, DC, USA, 2005. IEEE Computer Society.

[4] G. Fletcher, H. Sheth, and K. Borner. Unstructured peer-to-peer networks: Topological properties and search performance. In *AP2PC '04: Proceedings of the Third International Workshop on Agents and Peer-to-Peer Computing*, Lecture Notes in Computer Science, pages 14–27. Springer, 2004.

[5] M. R. Garey and D. S. Johnson. *Computers and Intractability; A Guide to the Theory of NP-Completeness*. W. H. Freeman & Co., New York, NY, USA, 1990.

[6] V. Gramoli, E. L. Merrer, and A.-M. Kermarrec. The GossiPeer homepage. `http://gossipeer.gforge.inria.fr`.

[7] T. Herman and S. Tixeuil. A distributed TDMA slot assignment algorithm for wireless sensor networks. Technical Report 1370, Laboratoire de Recherche en Informatique, Université Paris-Sud, Orsay, France, 2003.

[8] H. Jamjoom, S. Jamin, and K. Shin. Self-organizing network services. Technical Report CSE-TR-407-99, University of Michigan, 1999.

[9] L. Jia, R. Rajaraman, and T. Suel. An efficient distributed algorithm for constructing small dominating sets. *Distrib. Comput.*, 15(4):193–205, 2002.

[10] F. Kuhn and R. Wattenhofer. Constant-time distributed dominating set approximation. *Distrib. Comput.*, 17(4):303–310, 2005.

[11] E. Le Merrer, V. Gramoli, A. C. Viana, M. Bertier, and A.-M. Kermarrec. Energy aware self-organizing density management in wireless sensor networks. In *MobiShare '06: Proceedings of the 1st International Workshop on Decentralized Resource Sharing in Mobile Computing and Networking*, pages 24–29, New York, NY, USA, 2006. ACM.

[12] Q. Lv, P. Cao, E. Cohen, K. Li, and S. Shenker. Search and replication in unstructured peer-to-peer networks. In *ICS '02: Proceedings of the 16th international conference on Supercomputing*, pages 84–95, New York, NY, USA, 2002. ACM.

[13] L. Massoulié, A.-M. Kermarrec, and A. J. Ganesh. Network awareness and failure resilience in self-organising overlay networks. In *SRDS '03: Proceedings of the 22nd Symposium on Reliable Distributed Systems*, pages 47–55. IEEE Computer Society, 2003.

[14] T. Moscibroda and R. Wattenhofer. Efficient computation of maximal independent sets in unstructured multi-hop radio networks. In *Proceedings of the IEEE International Conference on Mobile Ad-hoc and Sensor Systems*, pages 51–59, October 2004.

[15] Peersim: http://peersim.sourceforge.net/.

[16] S. Ratnasamy, P. Francis, M. Handley, R. Karp, and S. Schenker. A scalable content-addressable network. In *SIGCOMM '01: Proceedings of the 2001 conference on Applications, technologies, architectures, and protocols for computer communications*, pages 161–172, New York, NY, USA, 2001. ACM Press.

[17] A. I. T. Rowstron and P. Druschel. Pastry: Scalable, decentralized object location, and routing for large-scale peer-to-peer systems. In *Middleware '01: Proceedings of the IFIP/ACM International Conference on Distributed Systems Platforms Heidelberg*, pages 329–350, London, UK, 2001. Springer-Verlag.

[18] K. Singh and H. Schulzrinne. Peer-to-peer internet telephony using sip. In *NOSSDAV '05: Proceedings of the International Workshop on Network and Operating Systems Support for Digital Audio and Video*, pages 63–68, New York, NY, USA, 2005. ACM.

[19] SIP: www.ietf.org/html.charters/sip-charter.html.

[20] I. Stoica, R. Morris, D. Karger, M. F. Kaashoek, and H. Balakrishnan. Chord: A scalable peer-to-peer lookup service for internet applications. In *SIGCOMM '01: Proceedings of the 2001 conference on Applications, technologies, architectures, and protocols for computer communications*, pages 149–160, New York, NY, USA, 2001. ACM Press.

[21] D. Stutzbach and R. Rejaie. Understanding churn in peer-to-peer networks. In *IMC '06: Proceedings of the 6th ACM SIGCOMM conference on Internet measurement*, pages 189–202, New York, NY, USA, 2006. ACM.

[22] C. Yang and J. Wu. Dominating-set-based searching in peer-to-peer networks. In *Proceedings of the Second International Workshop on Grid and Cooperative Computing*, Lecture Notes in Computer Science, pages 332–339. Springer, 2003.