Data Location Management Protocol for Object Stores in a Fog Computing Infrastructure

Bastien Confais, Benoît Parrein, Adrien Lebre

Data Location Management Protocol for Object Stores in a Fog Computing Infrastructure

Bastien Confais*, Benoît Parrein† and Adrien Lebre‡
* CNRS, LS2N, Polytech Nantes, rue Christian Pauc, 44306 Nantes, France
† Université de Nantes, LS2N, Polytech Nantes, rue Christian Pauc, 44306 Nantes, France
‡ Institut Mines Telecom Atlantique, LS2N, 4 Rue Alfred Kastler, 44300 Nantes, France

***Abstract*—Fog Computing infrastructures have been proposed as an alternative to Cloud Computing to provide low latency computing for the Internet of Things (IoT). But no storage solutions have been proposed to work specifically in this environment. Existing solutions, relying on a distributed hash table to locate the data, are not efficient because location record may be placed far away from the object replicas. In this paper, we propose to use a tree-based approach to locate the data, inspired by the Domain Name System (DNS) protocol. In our protocol, servers look for the location of an object by requesting successively their ancestors in a tree built with a modified version of the Dijkstra's algorithm applied to the physical topology. Location records are replicated close to the object replicas to limit the network traffic when requesting an object. We evaluate our approach on the Grid'5000 testbed using micro experiments with simple network topologies and a macro experiment using the topology of the French National Research and Education Network (RENATER). In this macro benchmark, we show that the time to locate an object in our approach is less than 15 ms on average which is around 20% shorter than using a traditional Distributed Hash Table (DHT).**

***Keywords*—Fog Computing, Data Location Management, Tree Overlay, Dijkstra's algorithm, DHT, IPFS, Grid'5000.**

I. Introduction

While largely adopted, Cloud Computing relies on few physical datacenters located far from the users. This model cannot satisfy the new constraints of the Internet of Things (IoT), especially in terms of latency and reactivity. The Fog Computing approach proposed by Cisco in 2012 [1] consists in deploying from micro to femto datacenters, geographically distributed at the edge of the network. Figure 1 illustrates a Fog infrastructure where the small datacenters are called "Fog sites". This approach avoids congestions by containing the network traffic but most importantly, the proximity to the users, enables the Fog infrastructure to provide low latency computing. Clients are potentially mobile and are always connected to the closest site in terms of network latency.

In this context, our ultimate goal is to develop a seamless storage solution where the clients can always access their data, regardless of the site they are connected to. In this work, we focus on the access pattern write-once, multiple-reads. This covers many use cases like symmetric CDN (Content Delivery Network) for consumer market where immutable content is shared from the edge of the network or providing a storage backend for Network Virtualization Functions (NFV) where immutable virtual machine images are relocated to follow users in their mobility [2].

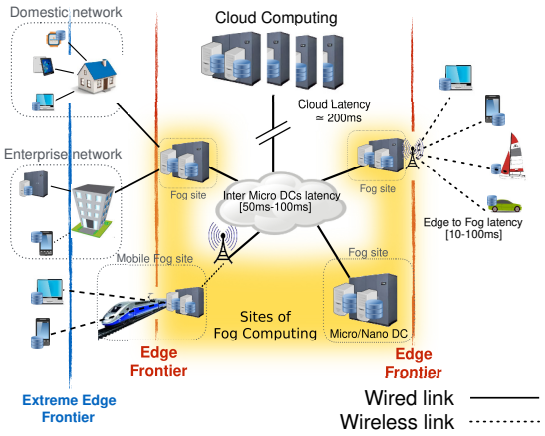


Fig. 1: Overview of a Cloud, Fog and Edge infrastructure. Each site can be composed of several "Fog nodes".

Object-stores writing data locally provide both locality and scalability, two useful properties for Fog Computing infrastructures [3]. Nevertheless, the records containing the location of all objects replicas are often stored without the locality constraint. At best, a Distributed Hash Table (DHT) is used. DHT have valuable properties *e.g.*, the position of location records is determined by a hash function which enables scalability and load balancing but the geographical dimension is not considered.

In a DHT-based solution, when an object is requested, a first site is reached to determine its location and then, the object is retrieved from a second site. Finally, a local replica is created to improve the performance for future accesses and the location record in the DHT is updated to reflect this new location. Objects are replicated where they are accessed but the location record is not relocated. We assume that reaching the site storing the location record may be longer than actually accessing the site storing the object replica.

In this paper, we propose a metadata management approach designed up to now for immutable objects. This approach in which the records storing the location of objects are replicated on the fly is inspired by the Domain Name System (DNS) [4] and by several implementations of Content Delivery Network (CDN) [5] protocols. Its main characteristic is to take into account the physical network topology in order to relocate the object's location records close along the routing path to the object's replicas. In this way, the servers are able to locate

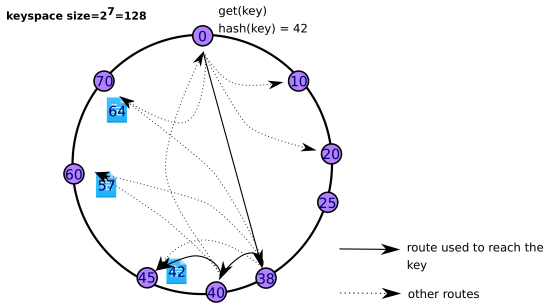


Fig. 2: Example of DHT routing to access the key with a hash equal to 42.

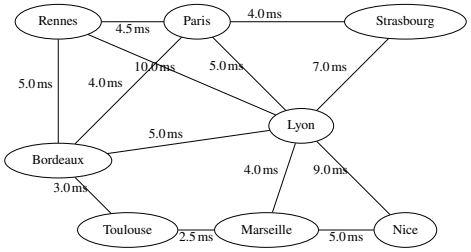


Fig. 3: Part of the French National Research and Education Network **physical** topology.

objects with a low latency and without sending requests far away in the network. We also present a method to generate a tree from a network topology by using the Dijkstra's algorithm and by modifying its cost function to adapt it to our iterative protocol.

In Section II, we describe why existing protocols and more precisely why DHTs are not adapted to Fog infrastructures. We provide some technical background to understand our protocol, and we list the assumptions we made in our work. In Section III, we describe our protocol and present different methods that can be used to build the tree. Then, in Section IV, we evaluate our approach on the Grid'5000 testbed both with micro and macro benchmarks. Finally, we discuss the overhead cost of our approach in terms of messages and how the dynamicity of Fog Sites can be handled in Section V. Some related works are presented in Section VI before concluding in Section VII.

II. Technical background

In this section, we first explain why the traditional approaches to store the location and to locate objects are not adapted to Fog infrastructures. Then we present how an approach designed from the Domain Name System (DNS) protocol can benefit from the Fog particularities. Finally, we list the assumptions we made in this work.

A. Expected properties for a localisation protocol

A good property for an object store working in a Fog infrastructure is to write objects locally [6]. Objects are stored on a node of the site the user writes and a mechanism such as a Distributed Hash Table (DHT), a gossip protocol or a hashing function is used to store the relation between each object's name and the location of its replicas [7]–[10].

In order to locate the objects and to store their location efficiently, we expect the location records to be found close to the site accessing the object and to benefit from data movements between the sites to improve the locality. When a site accesses an object and relocates it, other sites should be able to locate this new replica by reaching a closer site than the original one.

These expectations can be extended to 5 other properties: *(i)* the impact of the approach on the access times. It is evaluated according to the number of requests needed when an object is accessed or written. *(ii)* The second property is the amount of network traffic generated by the churn, when a site is added or removed. This property also depends on the number of location records that must be moved. *(iii)* The amount of network traffic exchanged to locate an object is another property we evaluate. *(iv)* The fourth property is the amount of knowledge about the topology that each node has. In some approaches, each node only knows its close neighbourhood while in others, each node has to know all the nodes that are part of the network. *(v)* Finally, the last property is the minimal number of location records that need to be stored for each object. In other words, some approaches require to store the location of every object on all the nodes in the network, while in others, the location of each object can be stored on only one node.

In a previous work, we focused on the object store called InterPlanetary FileSystem (IPFS) [10], that relies on a BitTorrent-like protocol to exchange objects between nodes and a DHT to store their location.

Distributed Hash Tables propose a trade-off between the amount of network traffic generated to locate an object and the knowledge of the topology by each node. Each key is stored on the node with the identifier immediately following the value of the hash computed from the key. Routing tables are also computed so that each node knows a logarithmic number of neighbours and can reach any key with a logarithmic number of hops. Figure 2 shows an example of a Chord DHT [11] where the hash function computes the value 42 for the key `key`. Therefore, the value associated to the key will be stored on the node with the identifier 45 reached in 3 hops from the node 0. According to the Chord routing protocol, each node forwards the request to the node of its routing table which has the closest identifier to the hash of the key looked for without exceeding its value. Each entry of the routing table is pointing to the node with the identifier immediately greater than $p+2^i$ where p is the identifier of the current node and i is varying from 0 to the log of the keyspace size. Based on these simple rules, the node 0 uses the route $0+2^5=32$ pointing to the node 38 because the next route $0+2^6=64$ is greater than the key with the hash 42 we are looking for. Similarly, the node 38, uses the route $38+2^1=40$ pointing to the node 40, and finally, the node 40 uses the route $40+2^1=42$ pointing to the node with the identifier 45.

In a previous work, we proposed to couple object stores like IPFS [10], to a Scale-Out NAS (Network Attached Storage) deployed independently on each site in order to prevent the

nodes from using the DHT when a client requests an object stored at its geographical location (in the site it is connected to) [12]. We showed this approach improves the performance particularly for small objects, for which the location time may be longer than the time to retrieve them. This approach also limits the amount of network traffic sent between the sites, and enables the clients to access data locally stored in case of network partitioning. Nevertheless, we still face this issue related to metadata management while accessing objects that are not locally stored. Because latency is the key element of Fog infrastructures, addressing it is essential.

Accessing the DHT to find the location of an object has many drawbacks. Soliciting a remote site to get the location of an object impacts the reading performance because the network latency to reach the site storing the location record may be high. The performance of the remote sites is also impacted because answering external queries increases the load of the servers. Finally, it prevents the system from working in case of network partitioning: if a site is isolated from the others, nodes cannot locate the objects stored on it.

We illustrate this with the Figure 3 that shows different points of presence of the French National Research and Education Network (RENATER) topology[1] but also their physical connections and their network link latencies[2]. We consider an object written in Paris and its location stored in Strasbourg (east of France). When the object is first read from Nice (south-east of France), the location of the object is found by accessing Strasbourg and the object is downloaded from Paris. But, when the object is then read from another city like Toulouse (south-west of France), we want the object to be located and downloaded from Nice. We do not want to reach either Paris or Strasbourg and want to benefit from data movements read after read.

Therefore, the DHT is not adapted for managing metadata in Fog infrastructures. If an object is stored in Paris, it may be possible for the location record to be stored on any site.

[1]https://www.renater.fr/

[2]https://pasillo.renater.fr/test/get_qosmetrics_resultsMULTICASTv4.php

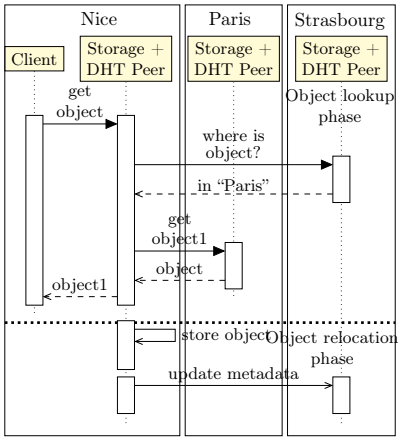


Fig. 4: Sequence diagram of network traffic when accessing an object remotely stored and when the location of the objects is stored in a DHT.

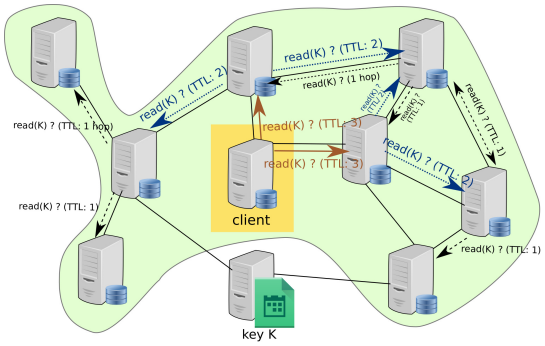


Fig. 5: Example of a flooding approach when the TTL specified is too low to reach the needed object.

Thus, in Figure 4, if a node located in Nice wants to access the object, it may have to reach a site like Strasbourg to locate it. Contacting an extra site to determine the location of an object can be a major issue when this site is reachable with a higher latency than the site that actually stores a replica of the object.

B. *Qualitative analysis of existing approaches*

Figure 7 presents a qualitative analysis of the existing solutions as discussed in the following.

1) Centralised metadata server: the use of a centralised metadata server does not scale and the need to reach a remote site to locate an object hugely impacts the access times. The main advantage of this approach is that all nodes only have to know the address of the metadata server to be able to locate objects and not the full network topology;

2) Flooding: this approach does not require to store the location of the objects. Each node forwards the requests to its neighbours until the request reaches the node storing the requested object. To avoid loops, requests are associated to Time to Live (*TTL*). The main drawback of this approach is the huge amount of time needed to reach a requested object and the large overhead cost of network traffic generated. Many nodes receive request for objects they do not store. There is also no guarantee to succeed in locating an existing object, as shown in Figure 5;

3) Gossip: this approach leads to store the location of every object on every node. Each time an object is written, its location is propagated in the neighbourhood until every node knows it. Once the location is propagated to all nodes, the main advantage of this approach is to be able to locate any object without sending anything on the network but the main drawback is that the location of each object is stored on all the nodes. Figure 6 shows an example of how location records are propagated throughout the network.

4) Hash function: applying a hash function on the name of the object is another approach. In this way, any object can be located without storing the location and requesting the network for it. But each node has to know the entire topology of the network to be able to associate the node address from the value computed with the hash function.

5) One-Hop Distributed Hash Table: a one-hop DHT is a mix between a hashing function and a gossip approach. Instead

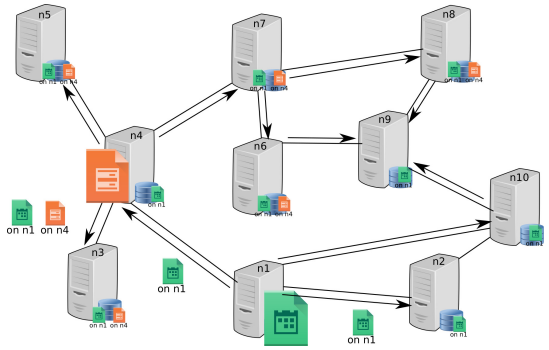


Fig. 6: Example of gossip propagation when the green object is located on node "n1" and orange object on node "n4". Nodes can receive several times the location record for a given replica (*i.e.*, "n9").

of sending the location of each object with a gossip protocol, the topology of the network is propagated to all the nodes by a gossip protocol. Each node sends to its neighbours the range of keys it is responsible for and builds a table containing these values for all the nodes of the network. Then, to locate an object, the hash of its name is computed and then the node responsible for the range of keys containing the hash value is directly reached. Once the network topology is gossiped to all nodes, the location process is similar to the use of a hashing function, that is why Figure 7(d) is similar to Figure 7(f).

For us, the best solution to store the location records does not impact the access times and it exchanges a small amount of network traffic when an object is located and when a site is added or removed from the network. For scalability reasons, it is also a solution that does not require each node to store the location of each object and to know the entire topology. In other words, it is a solution with the lowest values for the five properties.

C. The DNS protocol, an inspiration for our approach

Because neither the DHT nor any other traditional approach is able to provide physical locality and to relocate the records storing the location of object replicas close to where they are needed, we propose a protocol inspired by the Domain Name System (DNS). In the DNS protocol, a resolver who wants to perform a query, first requests a root node and if the root node cannot answer the query directly, it indicates to the resolver which server is the ablest to answer. This mechanism is similar to the hops performed in a DHT but with a major difference: it is possible to choose the node storing the location of object replicas instead of using a hash function. It is also possible to control the servers that a node needs to reach to perform a query which is a very interesting property in a Fog context. Also, reusing the cache mechanisms of the DNS to reduce the access times is something we want to take advantage of.

D. Assumptions

In our work, we make several assumptions. First, we work in a context where objects are immutable and therefore all replicas have the same content and are consistent. Second, the requests are sent in an iterative way *i.e.*,, the node that looks for an object sends the queries successively to the different nodes. Although the recursive approach is more performant, most DHT work in an iterative manner [13] for security reasons. It prevents the clients and nodes from waiting for a reply to a message that has been lost in the network. It also limits the impact on the load of the sites which are not the source of the request. We kept this approach in our protocol essentially for this last reason. Third, in this present work, we do not consider the dynamicity of Fog Sites. Nonetheless, this question will be discussed in Section V-B. Finally, we note that while performing a request, a Fog node does not discover the location of all the object replicas but only the closest one. In this way, the object is downloaded from a replica close to the site where the location record is found, providing locality in data accesses.

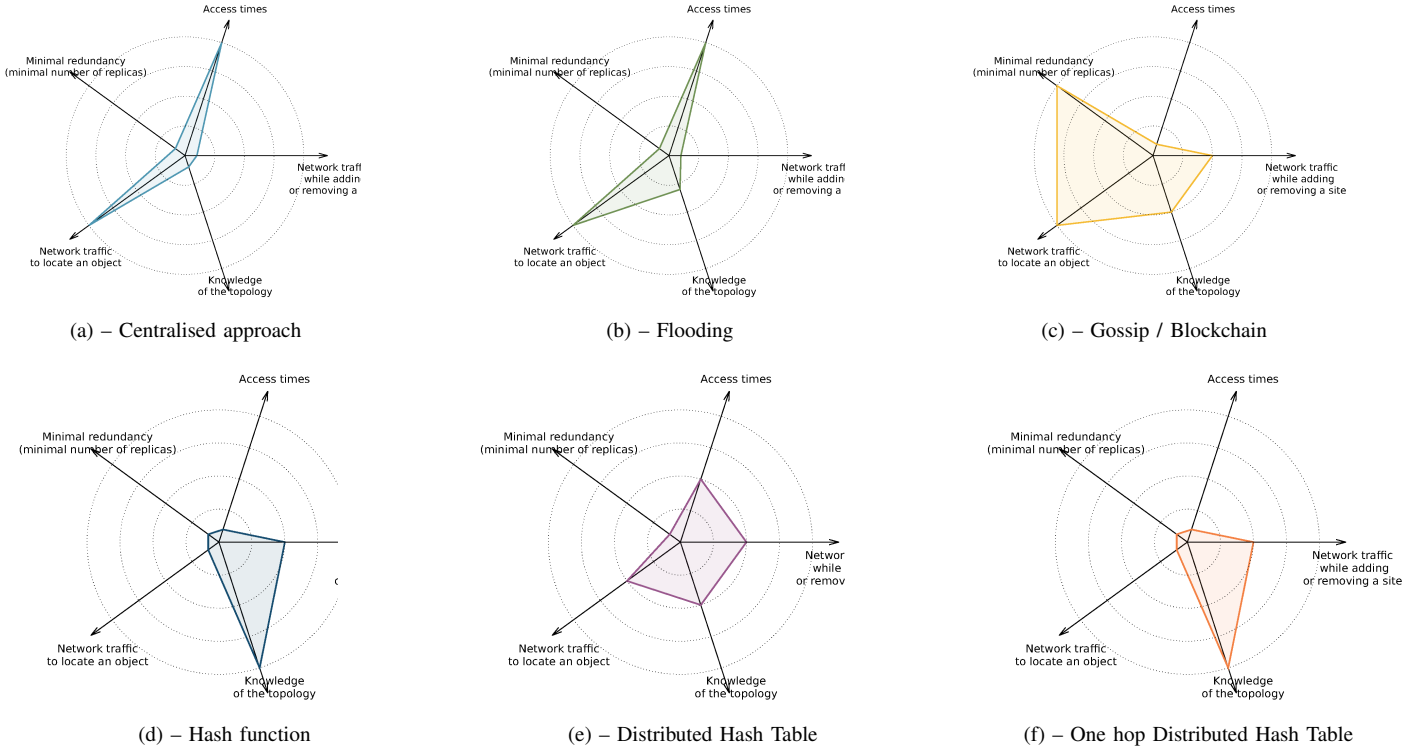
III. A TREE BASED APPROACH FOR OBJECT LOCATION

This section first describes how our protocol works, then proposes an algorithm to generate the tree we rely on and discusses the overhead cost of our approach.

A. Protocol description

We propose to distribute the records storing the location of object replicas inside a tree. Like in the DNS protocol, the different names are spread over different servers organised in a hierarchical way. The tree is composed of the different sites of Fog and is browsed from the current site to the root. If the location of an object's replica is not found at a given level, the parent node is requested. Contrary to the DNS where a resolver first requests the root node, our protocol uses a bottom-up approach. We consider that the tree is organised according to the physical topology including the links latencies in order to minimize the time needed to find the node storing a replica of the object. In other words, the parent of each node is close physically and looking for the location of the object by requesting it is faster than requesting the parent of the parent. Besides, it limits the network traffic to a small part of the topology.

Figure 8 shows the tree we computed from the network topology presented in Figure 3. How this tree was computed is explained later in the document but it will be used for sequence diagrams of Figure 9. Figure 8 also shows the metadata organisation. The edges between the nodes correspond to physical network links. Each node is able to answer all the requests for objects stored in their subtree, and more specifically, the root node located in Lyon is able to provide an answer for all the requests. The root node was chosen because of its central position in terms of latency and in the geographical topology (central east of France). As explained by Dabek *et al.*, network latency is dominated by the geographic distance [14]. We consider each site is composed of a "storage backend" and a "location server". The "storage backend" is in charge of storing the objects but also of retrieving them from other sites when they are not stored locally. The "location server" is responsible for storing the association between an object's name and the sites in its subtree, on which a replica is stored.



(a) – Centralised approach (b) – Flooding (c) – Gossip / Blockchain

(d) – Hash function (e) – Distributed Hash Table (f) – One hop Distributed Hash Table

Fig. 7: Star diagrams summarizing the properties for several approaches that can be used to locate data in a distributed system. The lower a value is, the more the property is compatible with a Fog Computing infrastructure.

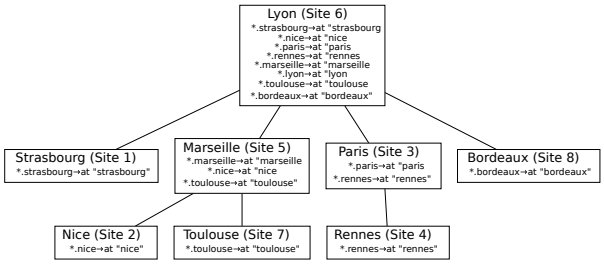


Fig. 8: Tree computed with our algorithm showing the initial content of the "location servers". Each site also has storage nodes and clients that are not represented.

Concretely, they store location records composed of an object's name and the address of a storage node storing a replica for this object. For a given object, a server stores at most one record per replica. Figure 8 also shows the initial content of the location servers. For instance, the `*.paris` record defines the default location of all the objects suffixed with `.paris`. The advantage of this special location record will be explained in the next paragraph. In the following sections, we use Figure 9 to explain how object replicas are created, accessed and deleted through this overlay tree. This figure shows the network messages exchanged by these nodes but also the physical path taken for routing them.

1) Object creation: Figure 9(a) shows that when a client writes an object, a suffix corresponding to the site where the object was created is added to the object's name. This leads to not update the location server as we see on the figure. With the wildcard delegation (*e.g.*, `*.paris`), location records are not updated when the object is created but only when additional replicas are added. In our Fog Computing vision, we assume that objects are mostly read from the site where they were created. This strongly reduces the amount of location records stored. Nevertheless, relying only on the suffix of an object to determine its location is not sufficient. A storage node in Paris should not be directly contacted when a user wants to access `objectX.paris`. Although we know a replica of the object is available in Paris, we must be sure there does not exist a closer replica in the network. Suffixes are only used here to limit the amount of update messages when new object replicas are created.

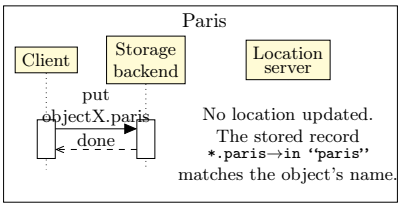
2) Accessing an object from a remote site for the first time: Figure 9(b) shows the reading process to obtain an object created in Paris from Nice. The client begins to request the local storage node and then, if the requested object is not stored locally, this node looks for the location of the object. The first location server requested is the local one, which is the closest. Then, in case of non-existent location, the storage node requests the parent of the location server (*i.e.*, Marseille) and so on, until one metadata server answers with the location. In the worst case, the location is found on the root metadata server (*i.e.*, Lyon). Once the optimal location found, the object stored in Paris is relocated locally, and because a new replica is created, the location record is updated asynchronously. The storage node sends an update to the location servers from the closest one to the one on which the location was found. Thus,

sites that are in the subtree of the updated "location servers" will be able to find this new replica in future reads. We note that, it could be possible to not use "location records" but to store directly object replicas within the tree, avoiding to first determine the location of an object before actually retrieving a replica. However, this strategy cannot be envisioned due to the storage space it would require.

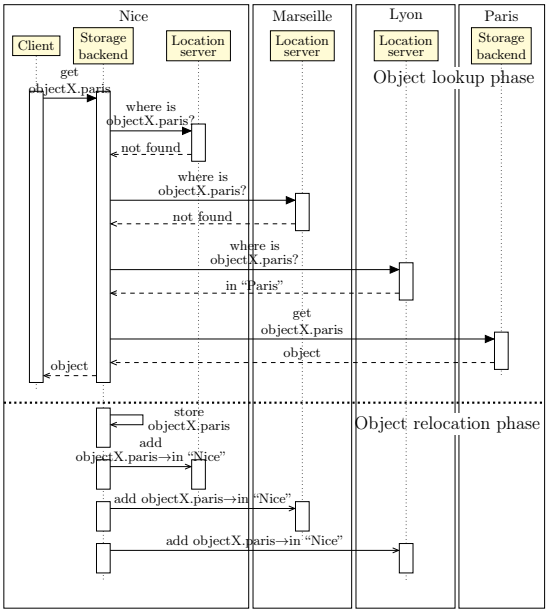
3) Accessing the object when several replicas are available: Figure 9(c) shows that when Toulouse requests the object created in Paris and previously relocated in Nice, the reading process is the same as previously described but the replica in Nice is accessed thanks to the location record found in Marseille. The root metadata server in Lyon is neither reached nor updated. We note that, despite a replica of the object was previously added in Nice, the object's name is not modified. Toulouse still looks for the object suffixed with `.paris` but now, thanks to the location record stored in the tree, it is able to access a closer replica stored in Nice. Therefore, suffix in object names does not have a meaning in the read process. Readers only need to know the site where the object was first written on, and not all the locations of the replicas.

This approach has several advantages. First, no network traffic is generated for objects that are written but never accessed. Secondly, the more sites access an object, the more replicas of data and location records there are. Also, in our approach, the object replica is always found in the subtree of the node where we found its location. Therefore, the closer the location, the closer the data. In other words, our approach enables the nodes to retrieve the closest replica (from the tree point of view).

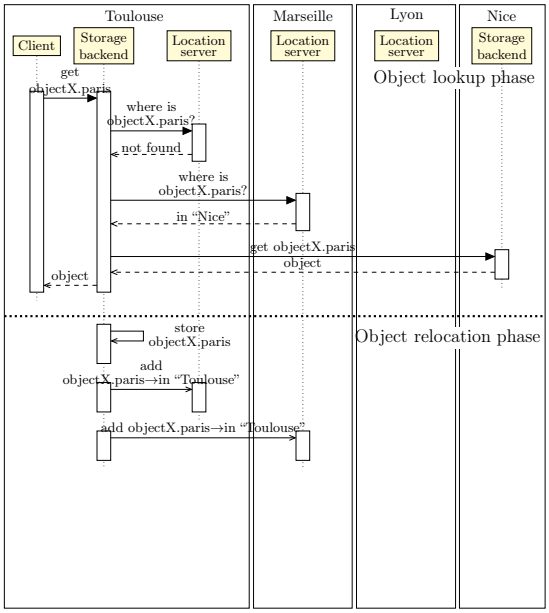
4) Deleting a single replica or removing an object entirely: To delete a single replica, the strategy we propose is to browse the tree from the site storing the replica to the first site that does not store any location record pointing to it. For instance to delete the replica located in Toulouse, the location record `objectX.paris→in "Toulouse"` is removed from the location servers of Toulouse and Marseille. The location records for this object that are stored in Lyon points to Nice and Paris and therefore, there cannot be a server higher in the hierarchy storing a replica pointing to Toulouse. In the case of removing the "master copy" of an object, we can browse the tree from the current node to the root node and to copy on each server of the path any location record stored in the root node that is not a wildcard for this object. For instance, deleting the replica stored in Paris leads to insert a record `objectX.paris→in "Nice"` in Paris. To delete an object entirely which consists in removing all its replicas, we propose to browse the tree from the root node and to follow and delete all the location records found for this object. Nevertheless, wildcard records can be followed but cannot be



(a) – Write an object on a local site (Paris).



(b) – Read the object stored in Paris from Nice.



(c) – Read from Toulouse the object previously read from Nice.

Fig. 9: Sequence diagrams of network traffic when a client writes an object in Paris and reads it first from Nice and secondly from Toulouse.

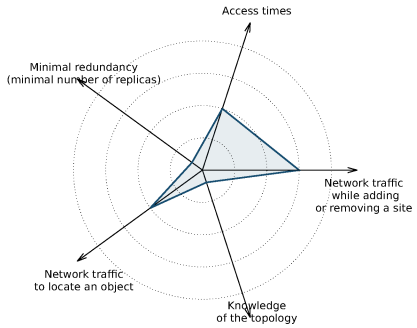


Fig. 10: Star diagram summarising the characteristics of our proposed approach.

deleted because they are also used for other objects.

To conclude this section, we have shown that our protocol is more adapted for Fog infrastructures than the DHT because the location is found along the physical path from the current node to the root node. Finally, in addition to reducing the lookup latency, the creation of location records enables the sites to locate reachable object replicas in case of network partitioning, increasing Fog sites autonomy. The properties of the proposed protocol are summarised in Figure 10. Our protocol limits the network traffic exchanged while locating an object and thus the impact on the access times. A second advantage is that it also limits the minimal number of replicas needed and the knowledge of the topology by the nodes. More precisely, in our protocol, location records are created only for objects accessed remotely and each site knows only its parent. We note that the amount of data to move the objects when a site is added or removed will be discussed in Section V.

B. Tree computation

As shown in Section III-A, the maximum latency to locate a replica (*i.e.*, the total latency to reach the root node) increases with the depth of the tree. For example in Figure 8, when a node in Nice finds the location of objects in Lyon, the total latency is the latency between Nice and Marseille to first request Marseille added to the latency between Nice and Lyon to reach the location record stored in Lyon (the link between Nice and Marseille is used twice, due to our iterative approach). Therefore, the tree used to find the location must be chosen with care in order to minimize the time to locate the objects.

The classical algorithm to compute the shortest paths from a source node to all the other nodes is the Dijkstra's algorithm [15]. This list of the shortest paths from a source node to any other node can be seen as a tree with the source node as root. We propose to reuse this algorithm to generate our tree but one of the drawbacks of the Dijkstra's algorithm is that the root node needs to be specified. In order to choose the "best" root node, we successively compute the tree with each node as source and select the one with the lowest weight. For instance, in Figure 11(a), the tree has a weight of $9.0 + 7.0 + 4.0 + 5.0 + 5.0 + (2.5 + 4.0) + (4.5 + 5.0) = 46$.

However, the cost function does not reflect the iterative way the servers are requested in our approach. For instance, in Figure 11(a), the weight of the Rennes site has a weight of 9.5 (*i.e.*, $4.5 + 5.0$). In our approach, the weight of this node should be equal to 14, that is $4.5 + (4.5 + 5.0)$ because nodes located in Rennes first request nodes located in Paris reachable in 4.5 ms and then, if the location is not found, request nodes located in Lyon and reachable in $4.5 + 5.0 = 9.5$ ms. Instead of using the original evaluation function of the Dijkstra's algorithm, we propose to evaluate the cost function shown in Equation 1 that considers the $depth$ of the nodes.

$$f_c = \left(\sum_{i=root}^{parent(node)} d\left(i, parent\left(\mathrm{i}\right)\right) \times depth\left(i\right) \right) + d\left(parent\left(node\right), node\right) \times depth\left(node\right) \quad (1)$$

The result of this modification is seen in Figure 11(b). Although this tree optimizes the total latency, it is very flat and most nodes directly access the root without benefiting from any relocation. In order to generate a deeper tree, we introduce a similar mechanism as proposed by Alpert *et al.* in the AHHK's algorithm [16] to relax the constraint. We connect a node through a specific link in the tree when the evaluated position of the node is deeper than its current one and when the total latency (as measured in Equation 1) is better or degraded by a factor smaller than c. Even if the latency to reach all ancestor nodes until the root is slightly increased, a deeper node has more ancestors and a greater chance to find a location record among them. Figure 11(c) shows the tree computed using this algorithm when $c = 1.2$ and used in the final macro benchmark.

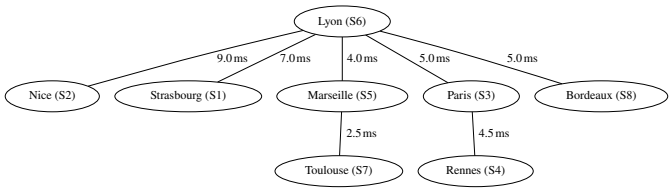
Although in the worst case, reaching the root node is longer than using the optimal tree (in Figure 11(b)), this relaxed tree provides a better average latency to locate any object. The average latency can be computed using Equation 2, showing how link latencies are weighted by the probability $p(j)$ to find the location record on the node j.

$$w_{\text{tree}} = \sum_{i \in nodes} \sum_{j=node}^{root} d(i,j) \times p(j) \quad (2)$$

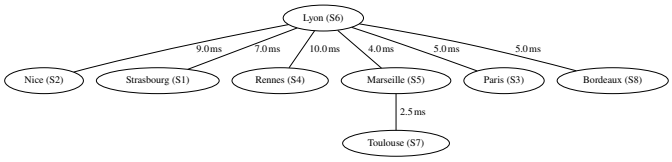
To compute the value $p(j)$, we consider a uniform workload among all the sites, *i.e.*, a given object has an equal probability to be accessed from any Fog site. For instance, with the tree shown in Figure 11(b), locating an object from Marseille requires on average $0 \times \frac{1}{7} + 4.0 \times \frac{6}{7} \approx 3.43\,ms$. We assume the object replica is not located locally in Marseille but on any of the 7 other sites because there is no need to use the location process to access a local replica. If an object replica exists in Toulouse (1 site among the 7), then Marseille already stores a location record and can locate the object in 0 ms. Otherwise, if the object replica is located on any of the 6 other sites, Lyon has to be reached with a latency of 4.0 ms. Equation 3 details the whole computation when we apply Equation 2 on the tree of Figure 11(b). It shows an average of 45.4 ms are required to locate an object.

$$\begin{aligned} w = &\left(9.0 \times \tfrac{7}{7}\right) + \left(7.0 \times \tfrac{7}{7}\right) + \left(10.0 \times \tfrac{7}{7}\right) + \left(4.0 \times \tfrac{6}{7}\right) + \\ &\left(5.0 \times \tfrac{7}{7}\right) + \left(5.0 \times \tfrac{7}{7}\right) + \left(2.5 \times \tfrac{1}{7}\right) + \left((2.5 + 4.0) \times \tfrac{6}{7}\right) \\ w \approx &\ 45.4 \end{aligned} \quad (3)$$

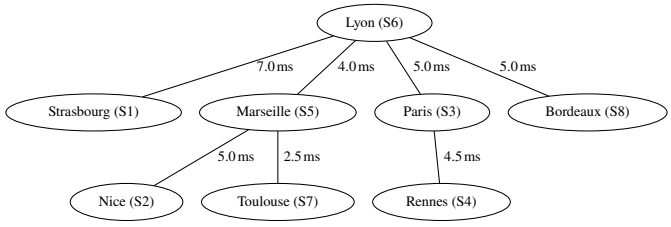
Equation 4 shows the same computation performed on the tree of Figure 11c. It enables us to conclude that the average



(a) – Dijkstra with the traditional cost function



(b) – Dijkstra with our cost function



(c) – Dijkstra with our relaxed cost function

Fig. 11: Trees generated from the French NREN physical topology using different cost functions.

latency to locate an object is lower with our tree (41.1 ms) built by relaxing the constraints than with the original one (45.4 ms).

$$\begin{aligned} w &= \left(7.0 \times \tfrac{7}{7}\right) + \left(4.0 \times \tfrac{5}{7}\right) + \left(5.0 \times \tfrac{6}{7}\right) + \left(5.0 \times \tfrac{7}{7}\right) + \\ &\left(5.0 \times \tfrac{2}{7}\right) + \left((5.0 + 4.0) \times \tfrac{5}{7}\right) + \left(2.5 \times \tfrac{2}{7}\right) + \\ &\left((2.5 + 4.0) \times \tfrac{5}{7}\right) + \left(4.5 \times \tfrac{1}{7}\right) + \left((4.5 + 5.0) \times \tfrac{6}{7}\right) \\ w &\approx 41.1 \end{aligned} \tag{4}$$

IV. Experimental evaluation

In this section, we compare experimentally our approach to a Kademlia DHT [17] used in the native version of the IPFS [10]. We first evaluate it with three different trees: (i) a balanced tree, (ii) a flat tree and (iii) a deep one. These three micro benchmarks enable us to fully understand how the time to get a location is impacted by the network topology. Then, in Section IV-C, we perform a macro benchmark on the topology of the French NREN (RENATER) to evaluate our proposal on a real network.

A. Material and Method

We performed our experiments using the InterPlanetary FileSystem (IPFS) object store [10] as a "storage backend". IPFS relies on a BitTorrent like protocol to exchange the objects between the nodes and on a Kademlia DHT to store the location of these objects. Each site of Fog is emulated by only one node collocating IPFS and the "location server". In order to mitigate our development efforts, we implemented "location servers" by using DNS servers. More specifically, we use BIND servers, configured as authoritative servers to store the records in flat text files (BIND default backend). In a experiment, we observed that the response time increased only after approximately 30 000 records were stored on a single server. Because we store fewer records, we consider that this backend does not impact our results.

The DNS servers provide the wildcard mechanism as well as a get/put protocol to request and to update the location records they store. We modified the routing mechanism used in IPFS to request these servers in a bottom-up manner rather than using the DHT, and to update the location records by sending Dynamic DNS (DDNS) messages. Because we do not rely on the original DNS lookup protocol, all servers are independent, enabling us to store several location records for a given object.

For a fair comparison, we removed the content based hash used in IPFS both in our version[3] and in the standard one that uses a DHT[4].

We use the topologies shown in Figure 12 and Figure 11(c) for the micro and macro benchmarks respectively. For each of the four topologies, objects are written on the first site (Site 1) and then they are accessed in parallel from other sites. In each read, each object is accessed from one and only one site that did not access it previously. In this way, we never read objects locally stored for which determining their location is not needed.

We measure the time to access a location record for each object, but we also measure the number of network links crossed to reach it (we call this metric the number of "hops").

We enabled IPFS to send several DHT requests in parallel. Different replication levels in the DHT are evaluated in order to be fair with our approach which creates new location records on the fly. We call "DHT k1". "DHT k2" and "DHT k3" a DHT with 1, 2 and 3 replicas respectively. We consider up to 6 replicas in our macro benchmark. The object repository of IPFS and the zone file of the DNS servers in our approach, are stored in a `tmpfs` in order to prevent any impact from the underlying filesystem.

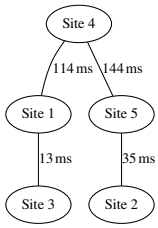
Tests are performed on the Grid'5000 testbed. Network latencies are emulated using the Linux Traffic Control Utility (`tc`). Network bandwidth between the sites is set to 1 Gbps. We use 1000 objects with a size of 4 KB each. We note that the size of objects has no impact on our results since we only measure the time to locate them.

Results are presented with a precision of 1 ms because it takes up to several seconds to locate some objects and only few milliseconds to locate others. All experiments were performed 10 times. For a better readability, standard deviations are not represented but are around 0.01 s.

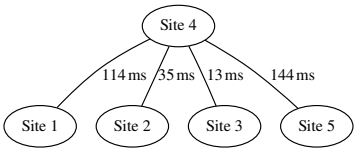
B. Micro benchmarks

In this section, we perform micro benchmarks using topology in Figure 12 to validate our experimental plan and to determine how our protocol behaves on simple topologies. These topologies are manually built from 5 sites of the Wondernetwork matrix of latencies [5].

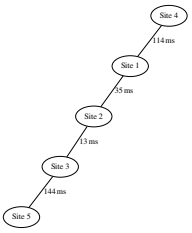
[3]https://github.com/bconfais/go-ipfs/tree/dns
[4]https://github.com/bconfais/go-ipfs/tree/dht_name_based
[5]https://wondernetwork.com/pings



(a) – Balanced tree topology.

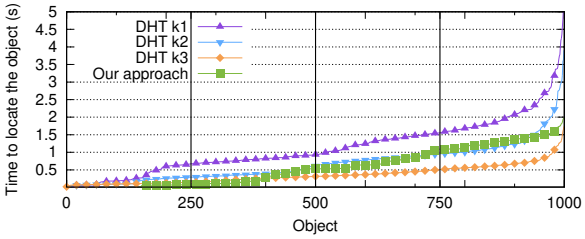


(b) – Flat tree topology.

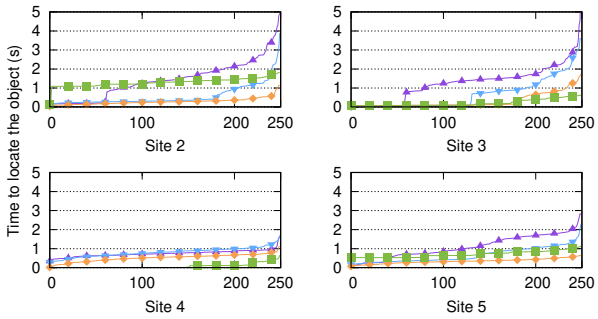


(c) – Deep tree topology.

Fig. 12: Micro benchmark topologies.

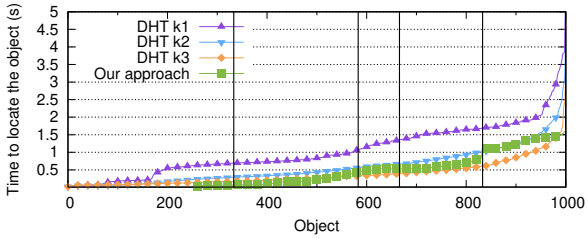


(a) – All sites

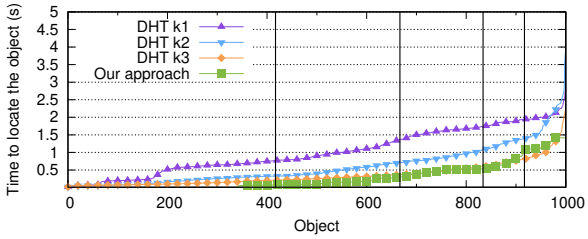


(b) – Per site

Fig. 13: Times to find the location of objects in the first read for all sites (a) and for each site (b). Objects are created on Site 1 and are sorted by the time to determine their location.



(a) – Second read



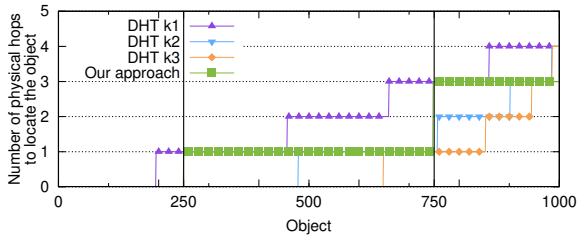
(b) – Third read

Fig. 14: Times to find the location of objects in the second read for all sites (a) and for the third read (b).

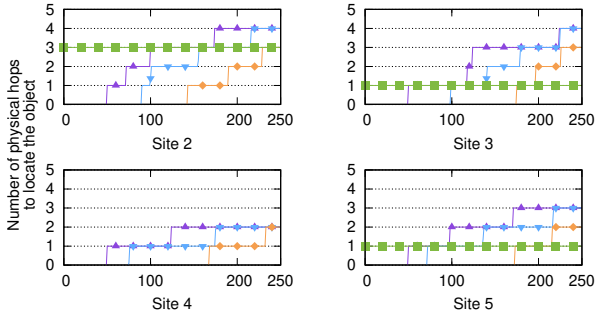
1) First topology, a balanced tree: we first evaluate our approach by performing the scenario with the tree given in Figure 12(a).

The time to locate objects is shown in Figure 13(a), both for the DHT and our approach when object are accessed for the first time. Objects are sorted from the time to locate them. It appears that locating an object with our protocol takes 1.982 s in the worst case which is faster than a DHT with only one replica (about 4.794 s) but longer than a DHT with 3 replicas (about 1.740 s). We nevertheless note that in the first read, our approach stores less than three replicas and it is logical to get higher access times than the DHT with 3 replicas. The comparison with "DHT k3" is not so really fair in a first read. Because objects are accessed in parallel, the time to locate the last object is also the time to locate all the 1000 objects. So, we can compute an average throughput of 504 objects located per second in our approach (vs 256 objects per second in the DHT with 2 replicas).

Vertical black lines of Figure 13(a) and following show the theoretical values delimiting groups of objects for which the location record is reached with the same network latency. In the first read, because each site locates objects with a different latency, we observe 4 different periods (one every 250 objects) separated by 3 different theoretical thresholds. For instance, before the first threshold we mostly observe objects read from Site 4 for which the location of object replicas is stored locally. The second group is almost composed of objects read from Site 3 for which the location of objects is found on Site 1, reachable in 13 ms (network latency between Site 1 and Site 3). From objects 500 to 750, we observe objects read from Site 5. Finally, after the last threshold, we observe the objects read from Site 2 which need to reach Site 5 and Site 4 to locate them. The non-linearity close to those lines means the observed result is what we expect. Because objects are sorted, theoretical thresholds do not exactly delimit what is happening site by site. To fully understand their individual behaviors, we split Figure 13(a) according to the site requesting each object. Figure 13(b) shows the time to locate an object is not the same for each site because the network latency to connect them is different. For instance, in the DHT, Site 4 cannot reach another node in less than 114 ms because it does not have a closer neighbor, whereas Site 3 can reach Site 1 in 13 ms. We do not observe non-linearity in our approach for a given



(a) – All sites



(b) – Per site

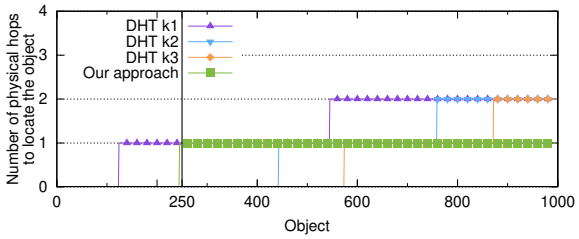
Fig. 15: Number of physical hops to reach the location record in the first read. Objects are sorted by the number of hops to determine their location.

site (Figure 13(b)) because each of them finds all the location records from the same location. We note that in Figure 13(a) the tail from objects 920 to 1000 is due to a bad parallelism of IPFS that we checked with sequential accesses (not presented here due to space constraints). This is also the reason why in our approach the times we measure are higher than the times we compute from the tree. In the tree, the worst access time is from Site 2 which reaches the root in $35 \times 2 + 144 = 214$ ms and thus can locate them in 428 ms (RTT latency to consider the time to send the request and to receive the reply) but it appears Site 2 can take up to 1.982 s to access the location record of an object. To remove this flaw, we next evaluate the performance in terms of hops rather than in absolute time.

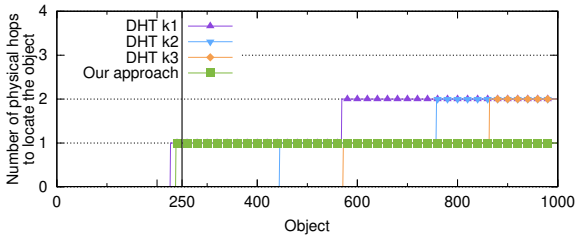
Figure 14 shows that in further reads, the time to locate the objects does not vary with the DHT but decreases in our approach that creates new location records when objects are accessed. For the third read, our approach becomes better than the approach using 3 replicas in the DHT: location records are close to the sites which need them instead of being spread uniformly. We need 1.417 s to locate the 1000 objects whereas the DHT needs 2.206 s in this case.

Because new location records are created according to object's access, these theoretical thresholds vary with the different reads. Therefore, for the second read, in Figure 14(a), location is found locally for 333 objects because Site 5 now stores the location for objects that were read from Site 2 in the first read. A similar observation is made on Site 2 for which objects are located from Site 5 instead of Site 4.

Figure 15(a) shows the number of physical hops to reach a location record for each object. Because of the iterative way the requests are sent, the number of hops in our approach



(a) – First read



(b) – Fourth read

Fig. 16: Number of physical hops to locate the objects in the first read (a) and fourth read (b) for the flat tree topology. Objects are sorted.

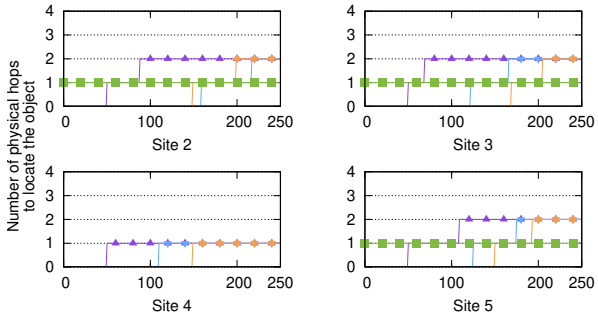


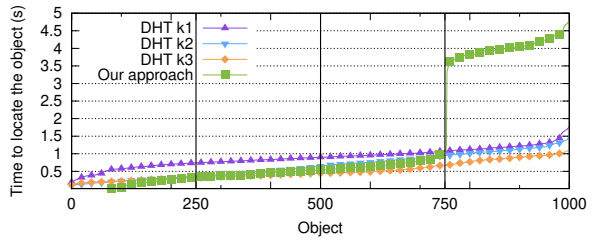
Fig. 17: Number of physical hops for each site to locate the objects in the first read for the flat tree topology. Objects are sorted.

can only be 0, 1 or 3 (a first request sent from Site 2 to Site 5 and another request sent from Site 2 to Site 4 that crosses two physical links). The result is similar to what we observed in Figure 13(a) but contrary to this figure, the number of physical hops is not impacted by the network latencies. Therefore, implementation biases that could speed up or slow down location times do not impact this result.

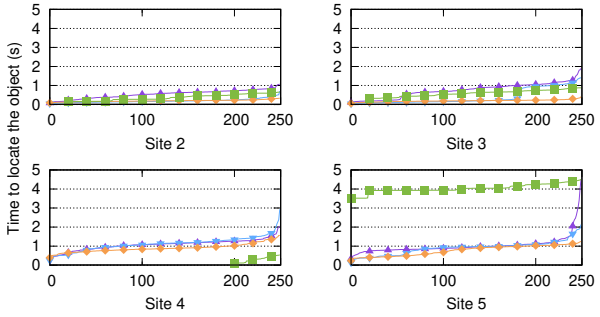
The conclusion of this experiment is that by requesting close nodes first and by creating new location records read after read, our approach provides better performance than the DHT.

2) Flat tree topology: the second topology we evaluate is shown in Figure 12(b). In this topology, our approach cannot benefit from the creation of new location records. When the location is not found locally, the root node is reached directly. Nevertheless, we show that even in this scenario, our approach outperforms the DHT. We focus on the number of hops in order to not consider the different possible latencies when a leaf node reaches another leaf node in the DHT.

Figure 16 shows the number of hops does not decrease between the first and the fourth read both in our approach and with the DHT. In our approach, object's location is always



(a) – All sites



(b) – Per site

Fig. 18: Times to find the location of objects in the first read for all sites (a) and for each site (b) for the deep tree topology. Objects are sorted by the time to locate them.

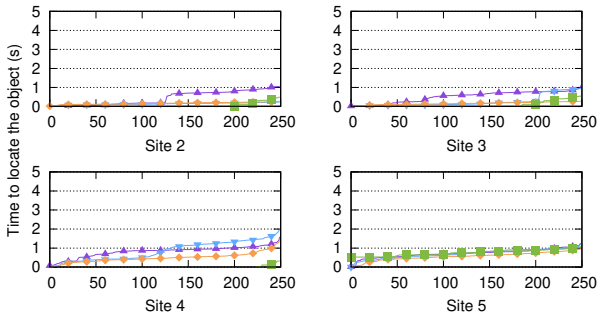


Fig. 19: Time to find the location of objects in the fourth read for each site in the deep tree topology.

determined by reaching Site 4. The theoretical threshold delimits the objects that are read from Site 4 for which location records are stored locally and the objects read from the other sites for which location is determined after one hop. If we look at the number of hops in Figure 17, we observe that our approach makes fewer hops than the DHT because in our approach, Site 4 knows where all the object replicas are and other sites request Site 4.

The conclusion of this experiment is that even when our approach does not benefit from the creation of new location records, it is still better than the DHT because objects are located with fewer hops. We however note a more important number of objects or peers may overload the root node. This highlights the importance of having a well-balanced tree.

3) Deep tree topology: the last topology we evaluate is the other extreme case in which all the sites are organised in a vertical tree as shown in Figure 12(c). We show that having a high-latency link close to a leaf node of the tree impacts negatively our approach, especially in the case where a lot of hops are needed to locate the objects. We observe in Figure 18(a) that reading the 1000 objects in 4.5 s in our approach makes it worse than the DHT which only needs 1.7 s. Figure 18(b) shows this result is due to Site 5, connected to the other sites with a high latency link (about 144 ms). In the DHT, this site finds the object location in one logical hops and thus, this network link is used only one time. In our approach, the link is used in the first hop to request Site 3, then in the second hop to request Site 2 and finally in the third hop to request Site 1. Because the high latency link is more solicited to retrieve the location, times are higher. This leads us to think to generate a tree in which deeper links, which are more solicited, have a lower latency. Figure 19 shows times are reduced in the last read because Site 5 locates all objects from Site 3, reachable in one hop.

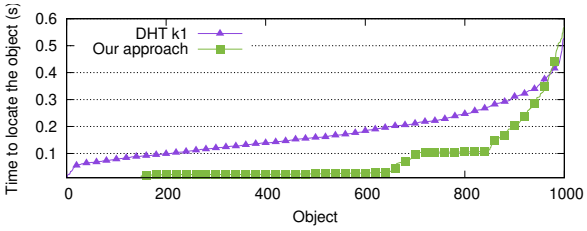
The conclusion of this experiment is that the high latency links close to a leaf node leads to degrade significantly the performance of some objects (objects 750 to 1000). Nevertheless, such a drawback can be mitigated when location records are added.

C. Macro benchmark

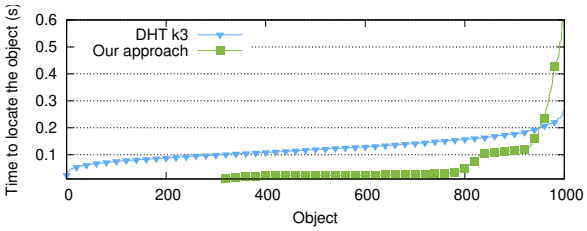
After performing micro benchmarks to understand how our protocol behaves, we perform a macro benchmark to see if our protocol can be used in real networks.

We consider the graph of a part of the French NREN shown in Figure 3. For our evaluation, we use the tree in Figure 11(c) that was computed as presented in Section III-B. For the DHT-based approach, we compute the shortest path (using the Dijkstra's algorithm) between each couple of nodes, so that each node can locate the objects with the best latency as possible. The consequence is the DHT benefits from optimal routing paths. We perform the same experiment as in the previous section. 1000 objects are written on Site 1 (Strasbourg) and are read successively from other sites.

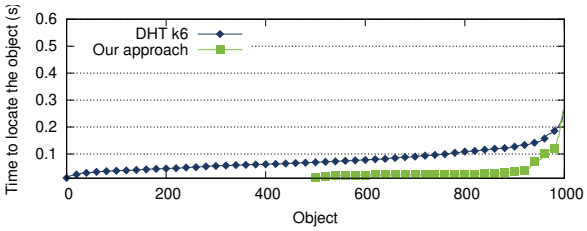
Figure 20 shows the times to access the location records in the first, the third and the seventh read. Because of the high number of sites, we performed the experiment with the DHT using 6 replicas for a fair comparison. We observe that for all reads, we have a better performance than the DHT, especially because in our approach, the closest nodes are requested first. In Figure 20(a), the gap we observe around 600 objects corresponds to the objects from which location record is accessed in 1 hop (accesses from Sites 5, 3 and 8) and objects that are located with 3 physical hops (Sites 2, 4 and 7 that have to reach their parent in 1 hop, and then, the root node in 2 extra hops). In the third read, shown in Figure 20(b), our approach becomes better than the DHT with 3 replicas thanks to the relocation, even if 5% of the objects (the last 50 objects) are read with a longer access time because they did not benefit from the relocation. For instance an object read in Lyon, then in Marseille benefits only from 2 replicas when the third read is performed. Contrary to this, an object read from Nice and then from Rennes benefits from five sites storing at least one location record in the third read. Finally, Figure 20(c) shows better performance in our approach because when location is not stored locally, it is requested on a close node.



(a) – First read



(b) – Third read



(c) – Sixth read

Fig. 20: Times to locate objects in the first, the third and the sixth read.

Figure 21 shows the total amount of time to locate the objects and to download them on the site where they are requested. It shows that better access times are achieved with our approach, not only because of our way of locating an object but also because sites always access the closest object replica. We also observe that the number of replicas in the DHT does not have a high impact on the total access times and thus, we represented the curve for 3 replicas only.

These experiments show that by limiting the amount of hops and by containing and reducing the network traffic sent between the sites, our approach enables the nodes to access the closest location record and reduces the time to access it. We also showed that our approach deployed with a real network topology still benefits from these characteristics.

V. Protocol limitations

In this section, we discuss some difficulties our protocol may face. We first evaluate its overhead costs in terms of the total number of messages exchanged while objects are accessed, and then we study how it may be possible to support the addition or the removal of a Fog site within the network without recomputing the whole tree, which is a costly operation.

A. Overhead costs evaluation

A drawback of our approach compared to a DHT is related to the number of messages sent to locate an object and to update the location servers. In this paragraph, we evaluate this message complexity as well as the complexity to compute the tree. In our comparison, we do not consider the updates to the local "location server" because it is reachable with a very low latency and thus neither generates inter-sites network traffic nor impacts the overall performance.

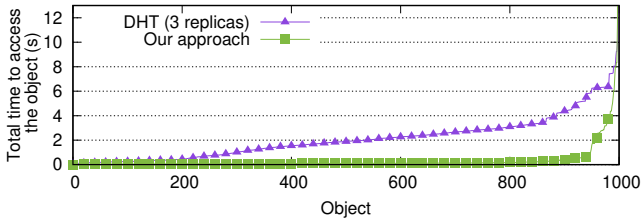


Fig. 21: Time to locate and to retrieve the objects in the sixth read.

From Figures 9(b) and 9(c), all location servers contacted when looking for the location of an object are updated. In the worst case, up to $depth(tree)$ update messages are sent, when the leaf of a degenerated tree accesses an object. In other words, the update complexity is $O(n-1)$ with n, the number of sites. With the DHT, the number of update messages is equal to the number of times the location of an object is stored (generally 3 for fault-tolerance), that is a constant ($O(1)$). Therefore, our approach is often more costly.

Because the number of update messages in our approach varies according to data movements, we need to consider the total number of update messages when an object is relocated on all the sites. The total number of update messages sent between the sites is equal to the number of edges in the tree (*i.e.*, the number of sites minus one). In the DHT with no replication, an update message is sent through the network for each site accessing the object except for the site that also stores the location of the object. Therefore, when objects are relocated on all the sites the number of update messages is equivalent both in our approach and in the DHT. We note that our protocol becomes less costly when we consider using replication within the DHT. Finally, the number of sites reached when a single replica is deleted is bounded by $O(depth(tree))$ because, in the worst case, the root node has to be reached, as explained in Section III-A4.

The complexity to build the tree is the same as the Dijkstra's algorithm which is $O(n^2)$ with n the number of sites. Yet, as we compute a tree with each node as root, the total complexity to build our tree is $O(n^3)$. In the following section, we discuss and propose some strategies to deal with such a complexity in the case of a dynamic topology, where the tree cannot be computed once and for all.

B. Handling Churn

Contrary to a DHT, our approach is not able to manage churn efficiently. In our approach the basic process when a new site joins or leaves the network consists in:

- Recomputing the whole tree using the algorithm previously presented to take into account this new node;
- Determining for each node the path from it to the root;
- Recreating wildcard delegations node;
- Moving location records to reflect the new tree topology.

This very costly strategy, cannot be used in practice and optimizations are welcomed. We note that, these actions should not be performed when a node is in a temporary failure but only when a site is created, closed, or when network latencies are changing.

Let us recall that some works have already discussed the dynamic shortest path tree problem which consists in rebuilding a tree without executing the Dijkstra's algorithm from scratch [18], [19] when the link latencies change. Unfortunately, a possible change in the root node of the tree have not been considered.

To add or remove a site efficiently, another strategy is to not modify the tree each time a site is added so that all the existing locations records are not moved. We only look for a parent for the new site by computing the average latency to locate any object for each possible position (shown in Equation 2). This reduces the complexity: only n different trees (where n is the number of nodes in the tree) are evaluated. This method has the advantage to enable a node to join the network with a linear complexity $O(n)$. A variant may consist in adding new sites only at the leaves of the existing tree in order to limit the number of trees evaluated. This approach also has the advantage of not recomputing all the wildcard delegations and all the location records as well: only adding the wildcard delegations for the new site is required.

Finally, a specific case can be considered for Fog sites that are mobile like public transportation. Because, their neighboring changes very often, it seems a good idea to consider these sites to be a leaf in the tree so that the tree is not to rebuilt entirely when their positions move in the topology.

VI. Related works

Many works propose to add locality in Distributed Hash Table (DHT) [20]–[23] but most articles only consider "routing locality". That is, only nodes with an identifier comprised between the identifier of the source node and that of the destination node can be reached during a lookup. This strategy enables nodes to access the closest replica. This strategy is proposed by many DHT using a Plaxton routing [24] to access the closest replica without requesting a node that is further. For instance, Zhao *et al.* proposed Tapestry [22], a DHT in which the time to reach a key is proportional to the distance to reach the node storing the key. The same idea was used by Wu *et al.* [25], Rowstron *et al.* [20] or by Locher *et al.* [23]. Yalagandula *et al.* [21] point out that routing locality is not sufficient, and they propose a DHT providing path convergence. The advantage is to enable administrative isolation between some sets of nodes, enabling sites to work in case of network partitioning, just like in our Fog Computing approach. As summarised by Castro *et al.* [26], the main drawback of those approaches is that the distance is computed in the node's identifier space that does not reflect the physical topology. To solve this problem, it is possible either to select the nodes identifiers or to change the DHT routing tables according to the network topology but these are difficult to implement.

Some approaches do not use a DHT. For instance, Koponen *et al.* [27] proposed DONA (Data Oriented Network Architecture), a distributed storage system able to locate the closest replica using an implicit tree. DONA uses the concept of Information Centric Network (ICN) so that no metadata lookup is performed: requests are routed thanks to the network layer. Using an ICN approach leads to several differences with our work. First, sites are at the edge of the network and thus, only at the leaves of the tree. Intermediate nodes are routers playing the role of metadata server and forwarding requests. This approach that does not require to first locate data before accessing it may be very useful in a Fog context. Nevertheless, ICN still faces many challenges such as dealing with topology changes or scalability issues. [28].

Let us also mention that comparing a DHT-based approach with the DNS has already been done several times [29], [30]. Pappas *et al.* [30] already showed the DNS approach has a better availability due to the hierarchical caches and also because the average path length to reach the record is shorter. But their experimentation is not strictly fair because they did not use the same number of nodes in the DHT and in the DNS approach. Moreover, this study does not consider locality and is not proposed in a Fog Context.

Finally, Content Delivery Networks (CDN) sometimes rely on a tree [31]. Contrary to our approach, data are always written at the root, simplifying the tree generation. Tree computation may also consider the allocation of each client to a specific site. Therefore, clients do not always request their closest site, leading to a better load balancing at the price of performance.

VII. Conclusion and Future work

In this article, we first described why locality in the location process is important. We showed that most existing protocols are not suited for a Fog environment, before introducing a new protocol relying on a tree and inspired by the DNS approach to store the location of objects. We also proposed to generate the tree overlay using a modified Dijkstra's algorithm adapted to the specificities of our protocol, before discussing the overhead cost of our approach and the challenges to make our protocol usable in a dynamic environment. Finally, we evaluated our approach on the Grid'5000 testbed. We showed with a micro and a macro benchmark that requesting close nodes first leads to better performance than the DHT, and we showed that creating new location records along the physical path improves further reading. Our protocol may benefit many write-once, read-many applications like Content Delivery Networks (CDN) or Network Virtualization Function (NFV). These use cases are particularly interesting because the location records are contained in the subpart of the network where the access are performed.

These scenarios imply to improve experimental evaluations by taking into account the workload of each site and by considering how our protocol behaves in case of network congestion and network partitioning. The power consumption of this proposal also need to be evaluated.

Many theoretical aspects should also be improved in future works such as supporting the mutability of the objects, maintaining the consistency among the replicas or dealing with

the dynamicity of the network topology. This last dimension is an obstacle for many approaches (like ICN) and relying on dynamic routing protocols or routing protocols for adhoc networks is something we may investigate.

REFERENCES

[1] F. Bonomi, R. Milito, J. Zhu, and S. Addepalli, "Fog Computing and Its Role in the Internet of Things," in *Proceedings of the First Edition of the MCC Workshop on Mobile Cloud Computing*, ser. MCC '12, 2012.

[2] R. Bruschi, F. Davoli, P. Lago, A. Lombardo, C. Lombardo, C. Rametta, and G. Schembra, "An sdn/nfv platform for personal cloud services," *IEEE Transactions on Network and Service Management*, vol. 14, no. 4, pp. 1143–1156, Dec 2017.

[3] S. Yi, C. Li, and Q. Li, "A survey of fog computing: Concepts, applications and issues," in *Proceedings of the 2015 Workshop on Mobile Big Data*, ser. Mobidata. New York, NY, USA: ACM, 2015.

[4] P. Mockapetris, "Domain names - concepts and facilities," RFC 1034, Network Working Group, Nov. 1987.

[5] D. Karger, E. Lehman, T. Leighton, R. Panigrahy, M. Levine, and D. Lewin, "Consistent hashing and random trees: Distributed caching protocols for relieving hot spots on the world wide web," in *Proceedings of the Twenty-ninth Annual ACM Symposium on Theory of Computing*, ser. STOC '97. New York, NY, USA: ACM, 1997, pp. 654–663.

[6] B. Confais, A. Lebre, and B. Parrein, "Performance analysis of object store systems in a Fog/Edge Computing Infrastructures," in *IEEE International Conference on Cloud Computing Technology and Science*, Luxembourg, 2016.

[7] G. DeCandia, D. Hastorun, M. Jampani, G. Kakulapati, A. Lakshman, A. Pilchin, S. Sivasubramanian, P. Vosshall, and W. Vogels, "Dynamo: Amazon's highly available key-value store," *SIGOPS Oper. Syst. Rev.*, vol. 41, no. 6, pp. 205–220, Oct. 2007.

[8] B. F. Cooper, R. Ramakrishnan, U. Srivastava, A. Silberstein, P. Bohannon, H.-A. Jacobsen, N. Puz, D. Weaver, and R. Yerneni, "Pnuts: Yahoo!'s hosted data serving platform," *Proc. VLDB Endow.*, vol. 1, no. 2, pp. 1277–1288, Aug. 2008.

[9] A. Lakshman and P. Malik, "Cassandra: A decentralized structured storage system," *SIGOPS Oper. Syst. Rev.*, vol. 44, no. 2, pp. 35–40, Apr. 2010.

[10] J. Benet, "IPFS - Content Addressed, Versioned, P2P File System," Protocol Labs, Inc., Tech. Rep., 2014.

[11] I. Stoica, R. Morris, D. Karger, M. F. Kaashoek, and H. Balakrishnan, "Chord: A scalable peer-to-peer lookup service for internet applications," *SIGCOMM Comput. Commun. Rev.*, vol. 31, no. 4, pp. 149–160, Aug. 2001.

[12] B. Confais, A. Lebre, and B. Parrein, "An object store service for a Fog/Edge Computing infrastructure based on ipfs and a scale-out NAS," in *2017 IEEE 1st International Conference on Fog and Edge Computing (ICFEC)*, Madrid, Spain, May 2017, pp. 41–50.

[13] F. Dabek, J. Li, E. Sit, J. Robertson, M. F. Kaashoek, and R. Morris, "Designing a DHT for low latency and high throughput," in *Proceedings of the 1st Conference on Symposium on Networked Systems Design and Implementation - Volume 1*. Berkeley, CA, USA: USENIX Association, 2004, pp. 7–7.

[14] F. Dabek, R. Cox, F. Kaashoek, and R. Morris, "Vivaldi: A decentralized network coordinate system," *SIGCOMM Comput. Commun. Rev.*, vol. 34, no. 4, pp. 15–26, Aug. 2004.

[15] E. W. Dijkstra, "A note on two problems in connexion with graphs," *Numer. Math.*, vol. 1, no. 1, pp. 269–271, Dec. 1959.

[16] C. J. Alpert, T. C. Hu, J. H. Huang, and A. B. Kahng, "A direct combination of the Prim and Dijkstra constructions for improved performance-driven global routing," in *1993 IEEE International Symposium on Circuits and Systems*, May 1993, pp. 1869–1872 vol.3.

[17] P. Maymounkov and D. Mazieres, "Kademlia: A Peer-to-Peer information system based on the XOR metric," in *International Workshop on Peer-to-Peer Systems*. Springer, 2002, pp. 53–65.

[18] P. Narváez, K.-Y. Siu, and H.-Y. Tzeng, "New dynamic spt algorithm based on a ball-and-string model," *IEEE/ACM Trans. Netw.*, vol. 9, no. 6, pp. 706–718, Dec. 2001.

[19] H. Chen and P. Tseng, "A low complexity shortest path tree restoration scheme for ip networks," *IEEE Communications Letters*, vol. 14, no. 6, pp. 566–568, June 2010.

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

[21] P. Yalagandula and M. Dahlin, "A scalable distributed information management system," *SIGCOMM Computing Comm. Rev.*, 2004.

[22] B. Y. Zhao, L. Huang, J. Stribling, S. C. Rhea, A. D. Joseph, and J. D. Kubiatowicz, "Tapestry: A resilient global-scale overlay for service deployment," *IEEE J.Sel. A. Commun.*, vol. 22, pp. 41–53, Sep. 2006.

[23] T. Locher, S. Schmid, and R. Wattenhofer, "equus: A provably robust and locality-aware peer-to-peer system," in *Sixth IEEE International Conference on Peer-to-Peer Computing (P2P'06)*, Sept 2006, pp. 3–11.

[24] C. G. Plaxton, R. Rajaraman, and A. W. Richa, "Accessing nearby copies of replicated objects in a distributed environment," *Theory of Computing Systems*, vol. 32, no. 3, pp. 241–280, Jun 1999.

[25] W. Wu, Y. Chen, X. Zhang, X. Shi, L. Cong, B. Deng, and X. Li, "LDHT: Locality-aware Distributed Hash Tables," in *2008 International Conference on Information Networking*, Jan 2008, pp. 1–5.

[26] M. Castro, P. Druschel, Y. C. Hu, and A. Rowstron, *Topology-Aware Routing in Structured Peer-to-Peer Overlay Networks*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2003, pp. 103–107.

[27] T. Koponen, M. Chawla, B.-G. Chun, A. Ermolinskiy, K. H. Kim, S. Shenker, and I. Stoica, "A data-oriented (and beyond) network architecture," *SIGCOMM Comput. Commun. Rev.*, vol. 37, no. 4, 2007.

[28] D. Kutscher, S. Eum, K. Pentikousis, I. Psaras, D. Corujo, D. Saucez, T. C. Schmidt, and M. Wählisch, "Information-Centric Networking (ICN) Research Challenges," RFC 7927, IETF, Tech. Rep., Jul. 2016.

[29] R. Cox, A. Muthitacharoen, and R. T. Morris, *Serving DNS Using a Peer-to-Peer Lookup Service*. Berlin, Heidelberg: Springer, 2002, pp. 155–165.

[30] V. Pappas, D. Massey, A. Terzis, and L. Zhang, "A comparative study of the dns design with dht-based alternatives," in *Proceedings IEEE INFOCOM 2006. 25TH IEEE International Conference on Computer Communications*, April 2006, pp. 1–13.

[31] A. Benoit, H. Larchevêque, and P. Renaud-Goud, "Optimal algorithms and approximation algorithms for replica placement with distance constraints in tree networks," INRIA, Research Report RR-7750, Sep. 2011.

Bastien Confais received his Master's degree in Computer Science in 2015 and his Ph.D. from the University of Nantes (France) last July. Thanks to his background in networking and in distributed systems, he is currently working on Fog Computing and more particularly on how to store data efficiently in such distributed environments. Over the last few years, he developed some skills to run experiments on testbed platforms such as Grid'5000 or FIT/IoT-lab.

Benoît Parrein (Member, IEEE) received the Ph.D. degree in Computer Science from the University of Nantes, France in 2001. He is currently Associate Professor at the University of Nantes (Computer Science department of Polytech school). He is member of LS2N laboratory (UMR CNRS 6004) dedicated to digital sciences. He is head of RIO research team dedicated to networks for the Internet of Things (IoT). His research interests are Fog and Edge computing, mobile ad hoc networks, robot networks and Intelligent Defined Networks (IDN). He co-authored more than 50 peer-refereed publications, contributed to 5 chapters in collective book and is coinventor of 3 patents.

Adrien Lebre is a full professor at IMT Atlantique, Nantes (France) and head of the STACK research group. He holds a Ph.D. from Grenoble Institute of Technologies and a habilitation from University of Nantes. His activities focus on large-scale distributed systems, their design, compositional properties and efficient implementation. Since 2015, his activities have been mainly focusing on the Edge Computing paradigm, in particular in the OpenStack ecosystem.