A Fog storage software architecture for the Internet of Things

Bastien Confais, Adrien Lebre, Benoît Parrein

▶ To cite this version:

Bastien Confais, Adrien Lebre, Benoît Parrein. A Fog storage software architecture for the Internet of Things. Advances in Edge Computing: Massive Parallel Processing and Applications, 35, IOS Press, pp.61-105, 2020, Advances in Parallel Computing, 978-1-64368-062-0. 10.3233/APC200004 . hal-02496105

HAL Id: hal-02496105
https://hal.science/hal-02496105

Submitted on 2 Mar 2020

November 2019

A Fog storage software architecture for the Internet of Things

Bastien CONFAIS [a] Adrien LEBRE [b] and Benoît PARREIN [c,1]

[a] *CNRS, LS2N, Polytech Nantes, rue Christian Pauc, Nantes, France*
[b] *Institut Mines Telecom Atlantique, LS2N/Inria, 4 Rue Alfred Kastler, Nantes, France*
[c] *Université de Nantes, LS2N, Polytech Nantes, Nantes, France*

Abstract. The last prevision of the european Think Tank IDATE Digiworld estimates to 35 billion of connected devices in 2030 over the world just for the consumer market. This deep wave will be accompanied by a deluge of data, applications and services. Thus, it is quite urgent to propose operational Fog architectures that support low latency, mobility (of users and possibly infrastructure) and network partitioning. In this chapter, we will detail such an architecture that consists in coupling an object store system and a scale-out NAS (Network Attached Storage) allowing both scalability and performance. Moreover, we provide a new protocol inspired from the Domain Name System (DNS) to manage replicas in a context of mobility. Finally, we discuss the conceptual proximity between Fog storage, Content Delivery Networks (CDN) and Name Data Networking (NDN). A complete experimental evaluation is done over the French cluster Grid'5000 in a second part of the chapter.

Keywords. Fog storage, data management, scale-out NAS, InterPlanetary File System (IPFS), Domain Name System (DNS), Content Delivery Networks (CDN)

1. Introduction

Proposed by Cisco in 2012 [BMZA12], Fog infrastructure consists in deploying micro/nano datacenters geographically spread at the edge of the network. Each small datacenter can be seen as a Fog site, hosting only few servers providing computational and storage resources reachable with a low latency. The new Internet of Things follows a hierarchical topology from the point of views of distance and power capabilities: Cloud facilities are the farthest elements in terms of network latencies but the ones that provide the largest computing and storage capabilities. Edge/Extreme Edge devices can benefit from local computing and storage resources but those resources are limited in comparison to the Cloud ones. Finally, Fog sites can be seen as intermediate facilities that offer a trade-off between distance and power capabilities of IT resources [BMNZ14,ZMK+15]. Moreover, Fog sites can complement each other to satisfy the needs between user's devices and Cloud Computing centers [BW15,DYC+15]. In a processing and storage as-

[1]Corresponding Author: Polytech Nantes, rue Christian Pauc, 44306 Nantes, France; E-mail: benoit.parrein@univ-nantes.fr. This work is the PhD work of Bastien Confais (defended in July 2018 at the University of Nantes, France) under the supervision of Adrien Lebre and Benoît Parrein.

pect, Fog sites are designed to process and store operational data that have a lifetime up to few days while Cloud infrastructures are designed to store historical data on several months [BMNZ14]. Thus, the architecture is both vertical (from the Cloud to the Edge) and horizontal (between Fog sites). This highly distributed infrastructure is necessary to support massive parallel IoT data streams from our point of view. The big question is what kind of distributed software is able to manage such infrastructure and for doing what?

Many use cases can benefit from such an infrastructure. For instance Bonomi *et al* [BMZA12] proposes to use the Fog Computing in a context of connected vehicles. The Fog collects metrics from them and makes the decision to stop the vehicles in a certain area if a pedestrian is detected on the road. Another use case is to deploy Network Virtualization Functions in the Fog so that the network functions follow the users in their mobility [BDL+17]. This should even be extended to a scenario where Fog Sites are mobile themselves, embedded in a bus or a train, to provide a low latency access to the users data during their travel. Finally, the Fog Computing can be used in Industry 4.0, where a large number of sensors uploads data to the fog, which is then handled by many users [TZV+11,WSJ17].

Figure 1 depicts such a hierarchy and some of the possible use cases. Each site hosts a limited number of servers that offer storage and computing capabilities. End-users devices (smartphones, tablets, laptops, ...) as well as the IoT devices (sensors, smart buildings, ...) can reach a Fog site with a rather low latency (noted L_{Fog}) lesser than 10 ms (latency of a wireless link). We consider the latency between Fog sites (noted L_{Core}) is up to 50 ms (mean latency of a Wide Area Network link [MTK06]). The latency to reach a Cloud platform (denoted L_{Cloud}) from the clients is higher (about 200 ms) and unpredictable [FGH14,SSMM14,ZMK+15,SSMM14]. We argue that storing data is a prerequisite to processing. Therefore, we need an efficient storage solution deployed in the Fog layer to handle the massive amount of data produced by the huge number of IoT devices. However, a single storage solution designed to work in this multi-site and low-latency context does not currently exist and our goal is to create it.

The first contribution of this work is to present the advantages and the drawbacks of different existing strategy to locate data in such a network. The second contribution is to propose to improve an existing storage solution by adding it a local Scale-Out NAS that enables it to locate locally stored data more efficiently. Finally, our last contribution is to propose a new protocol to locate data efficiently within a distributed network.

We particularly focus on the scenario of a symmetric Content Delivery Network, where data (*i.e.*, media documents) is produced at the edge of the network and is shared with the other sites. The main interest of such a usecase is that replicas are static replicas because the data is neither modified. This characteristic enables us not to consider the consistency.

In the following of the chapter, we will first describe what an ideal storage solution for Fog infrastructure is and present why existing solutions, developed for the Cloud architecture cannot be used. Then, we present our solutions to improve them.

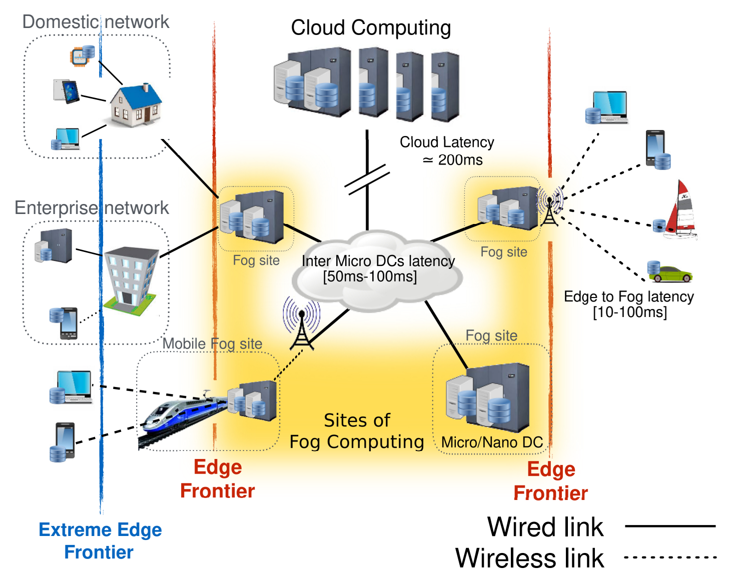


Figure 1. Cloud, Fog and Edge infrastructures for the IoT with related end-to-end delays.

2. Distributed storage for the Fog: an overview

In this section, we introduce the properties we expect from an ideal storage solution designed for a Fog infrastructure before explaining why the existing solutions and approaches used for Cloud Computing are not appropriated and must be revised.

2.1. Properties for an object store working in a Fog environment

Our goal is to create a unified storage solution for Fog architectures that can handle both a large quantity of data and a lot of accesses in parallel. This means each user and IoT device should be able to efficiently access all stored objects, regardless the site they are connected to. We previously established a list of 5 properties an ideal storage solution should meet [CLP16].

Data locality: in order to reduce the access times to the lowest value as possible, data should be written on the closest site of the user, which should be the site reachable with the lowest latency. Similarly, read data should be located on the site that restricts by consequence the forbidden remote accesses. Besides a high Quality of Service, the expected outcomes are also privacy and sustainability [BMNZ14].

Data availability: storage nodes are spread among a huge number of sites connected with non-reliable links. Therefore, an ideal storage solution should be fault-tolerant. Isolated failures should be managed inside the site they occur while an unreachable site should not impact the availability of the other sites. It is particularly the case with mobile Fog sites embedded in a bus or a train by example.

Network containment: Network containment consists in limiting the network traffic exchanged between the sites. This property has three aspects. First, only actions of the users should produce network traffic exchanged between the sites. Secondly, the access to a site should not impact the performance of the other sites. When a site is more solicited than the others, the other sites should not see their performance degraded. Finally, when users access data that is not stored on the site they are connected to, data should be located before being able to access it. Nevertheless, sites that do not participate in this exchange should not be contacted. In this context, the physical topology should be taken into account: exchanges using close sites, connected via a low latency link should be favoured. For instance, in case of document sharing between work and home, only the Fog sites located on the road between these two locations should be solicited.

Disconnected mode: the ability to access data stored locally on the site when no other site can be reached due to network failure is an essential property. Sites should be the most independent as possible. For instance, in a mobile Fog Site, users connected to it should access the data, even in case of network partitioning.

Mobility support: users are mobile and are always connected to the site reachable with the lowest latency. When the requested data is not stored on the local site, the data should be downloaded and cached on the current site to improve performance of future and predictable accesses.

Scalability: finally, an ideal storage solution for Fog architectures should be scalable to a large number of sites and a large number of clients.

To conclude, these properties can be understood with the Brewer theorem's in mind. The Brewer's theorem establishes that it is impossible to support simultaneously the consistency, the data availability (*e.g.*, the client gets a response in a finite time) and the partitioning of the network [GL02,Bre10]. Because, data availability and disconnected mode are in the list of the properties we want for an ideal storage solution, consistency is therefore not a property we can meet at all time and especially when the network is partitioned. This is also the reason why in this work, we consider all the data as immutable and where all replicas are static. In other words, we consider the access pattern to be write-once, read-many.

2.2. From Cloud storage solutions to Fog: the metadata management challenge

Because Fog nodes are geographically spread, the main challenge storage solutions will have to face to work in such an environment is how they locate the data.

Traditional distributed filesystems often rely on a centralised metadata server. As illustrated in Figure 2, the client first contacts a metadata server to locate the data and then reaches the node storing the replica. This approach is particularly used by High Performance Computing (HPC) solutions such as PVFS [CLRT00], Lustre [DHH+03] or IO intensive Scale-Out NAS such as RozoFS [PDÉ+14].

Nevertheless, this approach cannot provide the scalability required for large deployments characterized by a large number of storage nodes as well as a huge workload and cannot work in a distributed environment, where the centralised metadata server is not on the local site. To overcome this challenge, object stores propose a flat namespace, where each data is simply associated to a name. This reduces the amount of metadata to store and to access due to the absence of directories. As a consequence, the centralised metadata server is far less solicited and can be replaced by a more scalable but not so performant mechanism like a Distributed Hash Table.

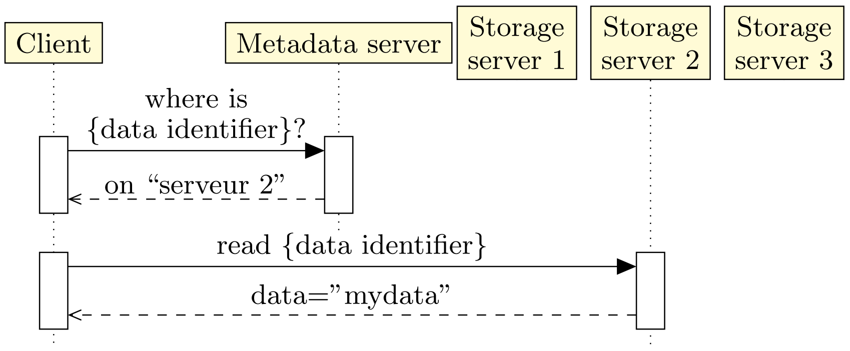


Figure 2. Sequence diagram showing how a client can access a data in a traditional distributed storage solution.

2.2.1. Approaches to locate data in a distributed manner

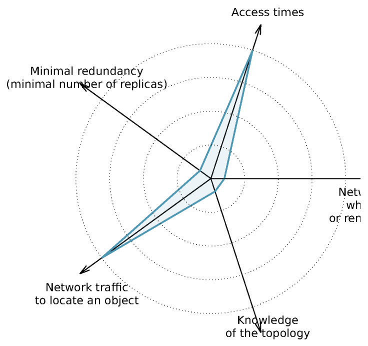
In this section, we present different approaches to locate data in a distributed manner that is a major functionality of any storage solution. We decline the 5 properties of Section 2.1 for a fog storage solution into 5 more specific properties that need to be met by the location process.

(i) Access times evaluate how close from the user the location records are stored. It is an extension of the “data locality” property to the location records.
(ii) Amount of network traffic while adding or removing a site and
(iii) Knowledge of the network topology evaluate how the location approach is scalable and reacts to the mobility of the sites themselves.
(iv) Amount of network traffic to locate an object is related to the above “network containment” property.
(v) Minimal redundancy corresponds to the “” property of the solution. The approaches that require to replicate the location records on all the nodes cannot scale to a huge number of objects.

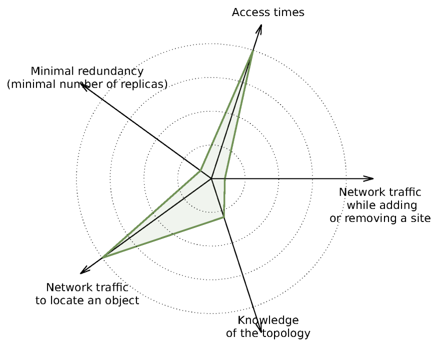
We evaluated 6 software solutions commonly used to locate data: a centralised metadata server, a flooding, a gossip, a hash function, a DHT and a one-hop DHT [DHJ$^+$07, CRS$^+$08,LM10,Ben14]. Figure 3 presents a qualitative analysis of these solutions in star diagrams.

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 advantages of this approach are 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. This also support network topology to change as long as the centralised metadataserver is not impacted;

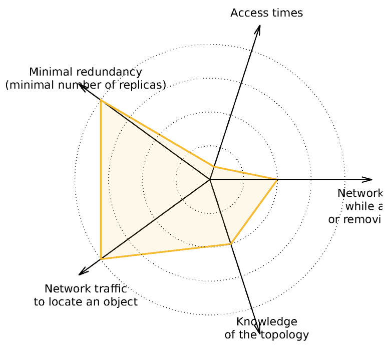
Flooding: this approach does not require to store the location of the objects. Each node forwards the requests to its neighbours until the request reaches the node storing the requested object. To avoid loops, requests are associated to a Time-to-Live (*TTL*). The advantage of this approach is not to require all nodes to know the network topology. And because the network topology is not know, the solution supports churn very well. However, the main drawback of this approach is the unpredictable amount of time needed to reach a requested object and the huge overhead of network traffic generated. Many



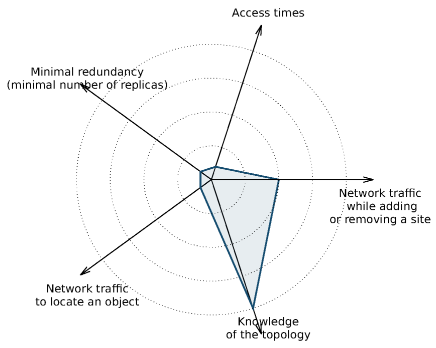
(a) – Centralised approach



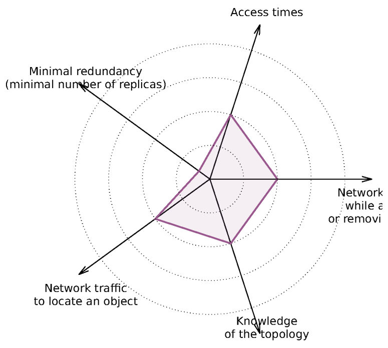
(b) – Flooding



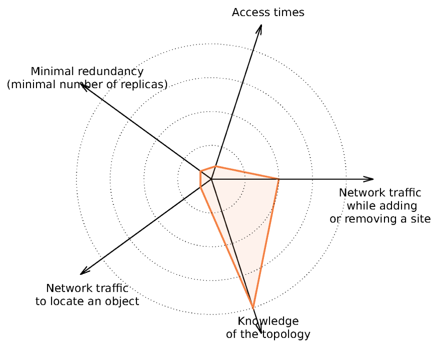
(c) – Gossip



(d) – Hash function



(e) – Distributed Hash Table



(f) – One hop Distributed Hash Table

Figure 3. Star diagrams summarising 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.

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 4.

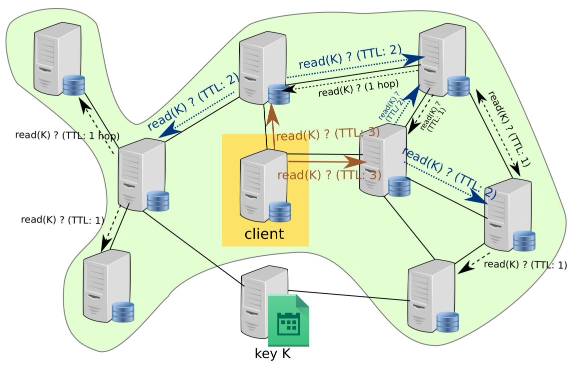


Figure 4. Example of a flooding approach when the TTL specified is too low to reach the needed object.

Gossip: this approach consists for each node to select a random neighbour and to send it all its knowledge about the location of objects. By repeating regularly this action, each nodes tends to know where all objects are located. It leads to store the location of every object on every node. Figure 5 shows an example of how location records are propagated throughout the network. 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. It also does not require the nodes to know the whole network topology but only their neighbouring. Because of this limited knowledge, managing churn can be done without propagating a lot of new information. Nevertheless, the main drawback is that the location of each object is stored on all the nodes, leading to increase the storage cost. We also should consider the huge amount of network traffic to propagate the location record of all pieces of data on all nodes prior to enable them to locate them without any extra exchange. In fact, instead of generating the network traffic when it is needed, it is generated in advance. This is why we think it is important to consider this background traffic as the network traffic required to locate a piece of data.

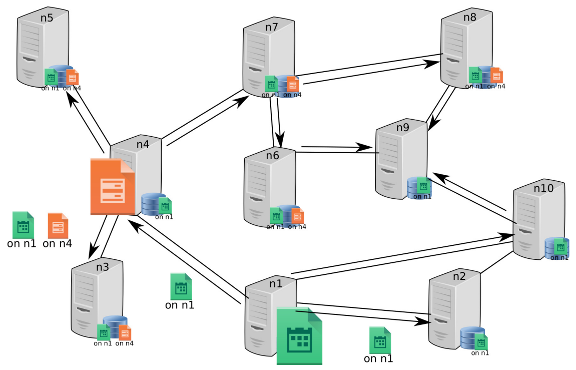


Figure 5. Example of gossip propagation when the green object is located on node "n1" and orange object on node "n4". Nodes can receive several times the location record for a given replica (*i.e.*, "n9").

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. The advantage is that locating can be done very quickly, without any network exchange. But, the drawbacks is not to explicitely choose where to store a specific piece of data. Also, 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. This also makes churn a new problematic because when the network topology change, there is no guarantee the existing mapping between hash values and nodes can be preserved. Therefore, churn requires to move some of the objects tored in the network. We note that consistant hashing is a particular hashing that minimizes the number of objects to move in case of churn.

Distributed Hash Table: a DHT proposes 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 6 shows an example of a Chord DHT where the hash function computes the value 42 for the key `key` [SMK+01]. 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. In the same way, 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.

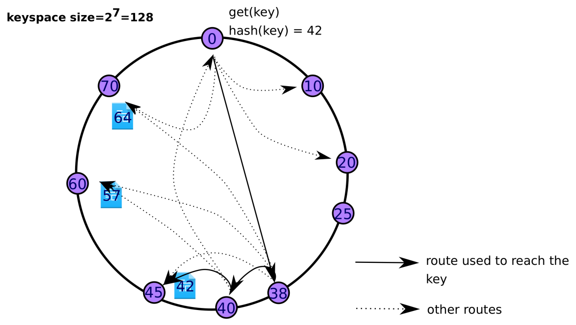


Figure 6. Example of DHT routing to access the key with a hash equal to 42

The advantage of the DHT is the to have the good tradeoff between the network exchanges required to locate an object and the amount of topology knowledge each node has. We also note that DHT enables churn by moving the minimal number of objects. Finally, the main drawback of the DHT is not to let user to choose where to store any piece of data.

One-Hop Distributed Hash Table: a one-hop DHT is a mix between a hashing function and a gossip approach. Instead 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 build 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. Because 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 3(d) is similar to Figure 3(f). The main drawback of such an approach is the location is replicated on every node which is costly in terms of storage space usage.

We argue the best solution to store the location records does not impact the access times, exchange a small amount of network traffic when an object is located and when a site is added or removed in the network. For 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.

2.2.2. Existing Cloud storage solutions

In this section, we propose to present different object stores designed to work in traditional Cloud infrastructure and evaluate how they behave in a Fog environment.

We focus on Rados [WLBM07], Cassandra [LM10] or InterPlanetary Filesystem (IPFS) [Ben14] that rely on the mechanisms we presented in the previous section to locate the data. Rados relies on a hashing function called CRUSH [WBMM06] to locate the data. An elected node is responsible for distributing a map describing the network topology as well as some placement constraints to all the storage nodes and clients. Then, each node is able to locally compute the location of each object wihtout any network exchange. The main advantage of Rados is to contain the network traffic thanks to its hash function but because this solution relies on a Paxos protocol to distribute the network topology description to all the nodes, it is not able to work in case of network partitioning. Also, moving exisiting stored data is not something easy making the support for mobility limited.

Cassandra is a solution originally developed by Facebook that uses a one-hop DHT maintained thanks to a gossip protocol. In Cassandra, each node regularly contacts another node chosen randomly to send its knowledge of the network topology. Then, like in Rados, objects can be located by each node without any other solicitation. The main difference between Rados and Cassandra is the way the map describing the topology is distributed among the nodes. Therefore, the advantages and drawbacks of Cassandra are the same as Rados, except that Cassandra does not rely on a Paxos protocol to distribute the description of the topology and therefore, can be used (is available according to the Brewer's theorem [Bre10]) in case of network partitioning.

Finally, IPFS is an object store relying on a BitTorrent protocol to exchange data between the nodes and on a Kademlia DHT to locate the nodes storing a replica for a given object. The advantage of this solution is to automatically create new replicas on the nodes they are requested and in case of network partitioning because each node has a limited knowledge of the network topology. But its disadvantages is to use immutable objects. In other words, objects in IPFS cannot be modified. We also not that the DHT spread to all nodes does not contain the network traffic.

Table 1 summarizes these points. Nevertheless, although all these solutions can deal with a huge number of servers, they are not designed to work in a multi-sites environment connected by high latencies and heterogeneous network links. Some distributed filesystems are designed to work in a multi-sites environment interconnected with Wide Area Network (WAN) links such as XtreemFS [HCK+08], GFarm [THS10] or GlobalFS [PHS+16] but because they are filesystems and not object stores and because they

	Purpose	Advantages	Disadvantages
Rados	Object store relying on a CRUSH hash function & a Paxos protocol	• Locate data without any network exchange (CRUSH)	• Cannot work in case of network partitioning (Paxos protocol) • Scalability difficulty (Paxos protocol) • Moving stored data with difficulty
Cassandra	Key/value store relying on a hash function & a Gossip protocol	• Place data without any network exchange • Works in case of network partitioning (Gossip)	• Moving stored data with difficulty
IPFS	P2P object store relying on a Bittorrent protocol & a DHT	• Automatically relocates data to new location: improve access times • Works in case of network partitioning	• The DHT does not contain the network traffic • Uses immutable objects

Table 1. Summary of the advantages and disadvantages of the presented storage solutions.

rely on a centralised approach, their performance is not what we can expect for Fog infrastructures.

2.3. How existing solutions fit the properties

In order not to develop a storage solution from scratch, we deployed the object stores previouly introduced (Rados, Cassandra and IPFS) a Fog infrastructure and evaluated their performance [CLP16] and how they met the properties we expect for a Fog storage solution. The complete analysis of this study is available in an article previously published by the authors [CLP16]. In this chapter, we just underline that none of the existing solutions meet the properties we defined for a Fog storage solution (as depicted in Table 2). For Rados, the use of Paxos algorithm makes data unavailable in case of service partitioning and limits its while for Cassandra, the lack of flexibility to move the stored object on the sites the users are moved to make access times higher. Finally, IPFS provides a native mobility because it does not have a placement strategy. Clients write where they want and the object stores keep track of where each data object is written.

	Rados	Cassandra	IPFS
Data locality	Yes	Yes	Yes
Network containment	Yes	Yes	**No**
Disconnected mode	**No**	Yes	**Partially**
Mobility support	**Partially**	**Partially**	Natively
Scalability	**No**	Yes	Yes

Table 2. Summary of Fog characteristics *a priori* met for 3 different object stores.

In the following, we propose to use the IPFS object store as a starting point and we propose some modifications to improve its behaviour in a Fog environment so that all the properties are met. We chose to start from IPFS because the mobility support is an essential property that cannot be added without modifying the placement strategy at the core of the storage solutions.

3. Designing a scalable and efficient Fog Layer

In a Fog Context, the inter-sites network exchanges are costly. It is important to limit them, not only to reduce the access times but also to enable the system to work in an environment with huge network latencies (with the extreme case of infinite latency due to network partitioning). In an ideal case, communication between the sites should only occur when an object is requested from a remote site.

In the following sections, we focus on the IPFS object store to adapt it to Fog environment. The goal of the first modification presented in Section 3.1, is to contain the network traffic while users access to locally stored objects. To that end, we proposed to couple IPFS to a Scale-out NAS solution.

The second modification consists in storing the location of the objects in a tree mapping the physical network topology to contain the network traffic in the data location process. This new protocol is covered in Section 3.2 of this chapter.

3.1. Reduction of inter-sites network traffic for locally stored data

In this section, we first present why the DHT used by IPFS to locate an object is not able to contain the network traffic. Secondly, we describe our approach consisting in coupling IPFS to a Scale-Out NAS that enables users to access objects locally stored on the site they are connected to without sending any inter-site network request.

3.1.1. Problem description

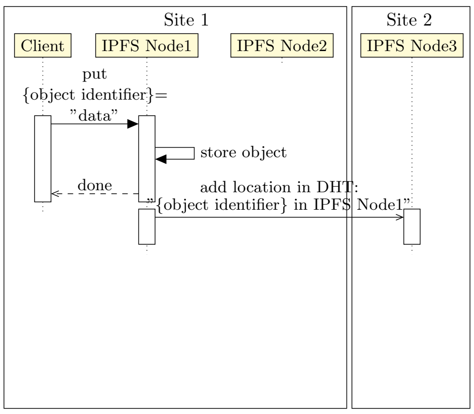
To understand the DHT limitation, we propose to detail the network exchanges we observe in IPFS. The writing and reading process are shown in the sequence diagrams of Figure 7. Figure 7(a) shows the creation of a new object : the client sends the object to any node located on the closest site (in terms of network latency). The object is stored locally and the DHT storing the location of each object is updated. Because the DHT does not provide any locality, the location of the object can be stored on any node of the network. In our example, the Node 3 located on Site 2 stores the location of an object that was written on Site 1. This node was determined thanks to a hash function on the object identifier.

Figure 7(b) shows what happens when a client reads an object stored on the local site. Each time a node receives a request for a particular object, it checks if the object is not stored on its hard drive thanks to a bloom filter for instance. If it is not the case, *(i)* a hash is computed according to the object name, *(ii)* the node storing the location is contacted, *(iii)* the object is downloaded thanks to a BitTorrent like protocol, *(iv)* a local replica is created while the object is forwarded to the client and *(v)* the DHT is updated to reflect the existence of this new replica. Finally, Figure 7(c) describes the protocol when an object is requested from a remote site: when the client moves from one site to another. We nevertheless precise that in all the situations, the client contacts a node located on the closest site (in terms of network latency).

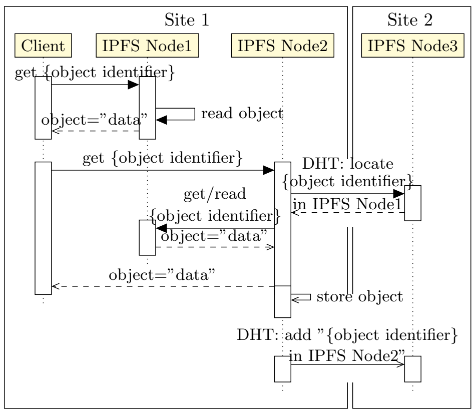
In other words, IPFS and more specifically the DHT it uses, does not take into account the physical topology of the Fog infrastructure. Each server is considered as independent rather than being part of a cluster of computers located on a on same site. When a new object replica is created, either by a user or by the relocation process, the other IPFS nodes of the site are not informed of the existence of this new replica and relies on the DHT to locate it. This is illustrated by the second read of the Figure 7(c). While the object is already available on the Node 4 of the Site 3, the Node 5 must contact the DHT to locate it. This has two main drawbacks. First, such a process increases the access times and network traffic exchanged between the sites. Secondly, the clients cannot access all the objects stored on the site when the site is disconnected to the other (network partitioning) because the location of objects cannot be found. IPFS, provides an efficient way to access data thanks to the BitTorrent protocol but does not take into account the specificities of a Fog Computing architecture.

3.1.2. A Fog storage software architecture

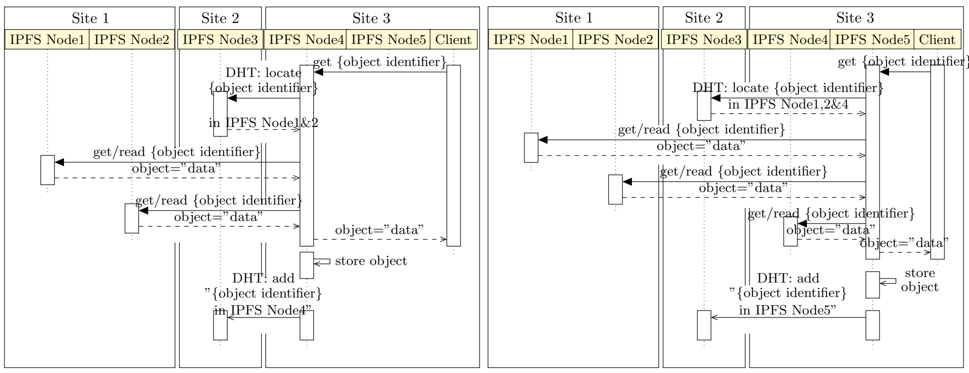
In order to improve IPFS in a Fog Computing environment, we propose to deploy a distributed filesystem independently on each site, such as a Scale-Out NAS. This Scale-Out


Site 1
Site 2
Client
IPFS Node1
IPFS Node2
IPFS Node3
put {object identifier}= "data"
store object
done
add location in DHT: "{object identifier} in IPFS Node1"

(a) – Writing of an object.



(b) – Reading of an object stored on the local site.


Site 1
Site 2
Site 3
IPFS Node1
IPFS Node2
IPFS Node3
IPFS Node4
IPFS Node5
Client
get {object identifier}
DHT: locate {object identifier}
in IPFS Node1&2
get/read {object identifier}
object="data"
get/read {object identifier}
object="data"
object="data"
store object
DHT: add "{object identifier} in IPFS Node4"
DHT: locate {object identifier}
in IPFS Node1,2&4
get/read {object identifier}
object="data"
get/read {object identifier}
object="data"
get/read {object identifier}
object="data"
object="data"
store object
DHT: add "{object identifier} in IPFS Node5"

(c) – Reading of an object stored on a remote site (by requesting a local node: IPFS Node4 or IPFS Node5).

Figure 7. Sequence diagrams showing (a) the writing process, (b) the reading of an object stored on the local site and (c) on a remote site with the IPFS solution.

NAS is used as an underlying storage solution for all the IPFS nodes of the site, as illustrated in Figure 8. This approach enables IPFS nodes to access all objects stored by the other nodes of the site, while requiring very few modifications in the source code. This coupling is possible because each IPFS node internally stores each object in a file. Instead of accessing a file on the local filesystem, IPFS will access the distributed filesystem provided by the Scale-Out NAS.

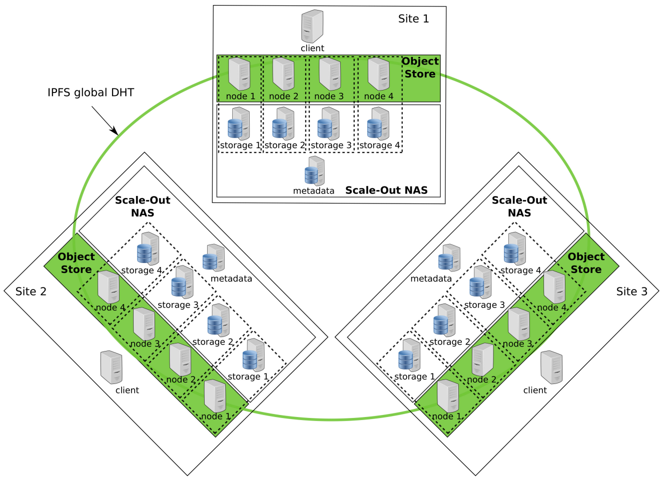


Figure 8. The Fog software architecture combining an object store (IPFS) and a Scale-Out NAS (like RozoFS).

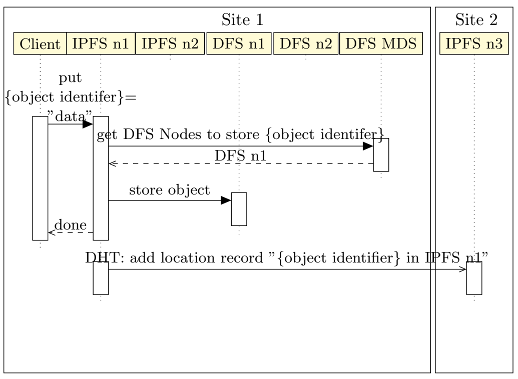
Coupling an object store with a Scale-Out NAS is not a new idea and many proposals have already been made. For instance, Yahoo developed Wlanut [CDM+12], an object-store relying on different filesystems such as Hadoop Filesystem (HDFS). However, our approach is a bit different because the underlying Scale-Out NAS is not a global filesystem but a distributed filesystem local to each site. IPFS is only a "glue" to create a global namespace through the different sites. A similar approach was already proposed in Group Base FileSystem (GBFS) [LBB14] where different filesystems deployed at different locations are aggregated into a common namespace. This solution was not efficient because all accesses starts by requesting the server responsible for the root directory leading to a huge workload. The main difference in our approach is that we rely on an object store and therefore there is no root directory to act like a bottleneck.

Finally, the interest of using a Scale-Out NAS such as RozoFS [PDÉ+14] or GlusterFS [DO13] is the capability to add nodes on the fly, increasing not only the storage space but also the performance of the system because the requests are spread uniformly between the different nodes.

Figure 8 shows the modification we made in the deployment of IPFS while Figure 9 shows through sequence diagrams the interactions between the different components.

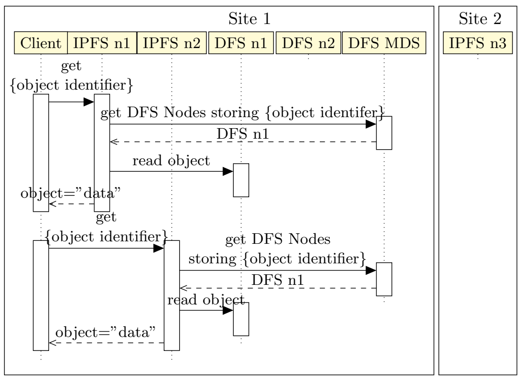
We observe there is no change in the protocol when an object is created, except that instead of storing the object directly on the node, it is sent to the distributed filesystem.

The main changes occur when a client wants to access an object. Figure 9(b) shows this changes: thanks to the Scale-Out NAS, all IPFS nodes of a site see the objects manipulated by the other nodes on the same site. As a consequence, regardless the IPFS node requested, when the node checks if the object is stored locally, it requests the Scale-Out NAS and finds it as if the object was written by itself. Contrary to the original conception

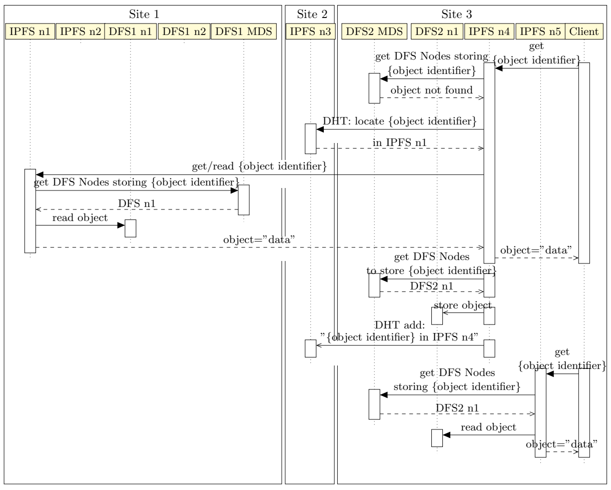


Site 1
Site 2
Client
IPFS n1
IPFS n2
DFS n1
DFS n2
DFS MDS
IPFS n3
put {object identifer}= "data"
get DFS Nodes to store {object identifer}
DFS n1
store object
done
DHT: add location record "{object identifier} in IPFS n1"

(a) – Writing.



(b) – Reading of an object locally stored.



Site 1
Site 2
Site 3
IPFS n1
IPFS n2
DFS1 n1
DFS1 n2
DFS1 MDS
IPFS n3
DFS2 MDS
DFS2 n1
IPFS n4
IPFS n5
Client
get {object identifier}
get DFS Nodes storing {object identifier}
object not found
DHT: locate {object identifier}
in IPFS n1
get/read {object identifier}
get DFS Nodes storing {object identifier}
DFS n1
read object
object="data"
object="data"
get DFS Nodes to store {object identifier}
DFS2 n1
store object
DHT add: "{object identifier} in IPFS n4"
get {object identifier}
get DFS Nodes storing {object identifier}
DFS2 n1
read object
object="data"

(c) – Reading of an object stored on a remote site.

Figure 9. Sequence diagrams showing the reading and writing processes when IPFS is used on top of a distributed filesystem (DFS) deployed on each site independently.

of IPFS, there is no need to request the DHT to locate objects that are locally available on a site.

In a similar way, Figure 9(c) shows the protocol when a client wants to access a data stored on a remote site. As usual, the client sends its request to any IPFS node of the site. The node checks in the distributed filesystem if the object exists and because it cannot find it, it requests the DHT to locate a replica. The object is then downloaded and a new replica is stored in the local Scale-Out NAS before the DHT is being updated. Future accesses performed from this site will be answered without any communication with the other sites, regardless the requested IPFS node .

This enables IPFS to meet the “network containment” property in case of local access but also to enable each site to work independantly in case of network partitioning. This is illustrated in Table 3. We finally have to remind the objects stored by IPFS are immutable, thus, we are not concerned by checking if the local replica stored in the local Scale-Out NAS is the most recent version of the object or if updated version can be found on another site.

To summarise, our approach limits the network traffic for accessing locally stored objects because when a client requests an object which has one of its replica available on the local site, the DHT is not used. In the other situations, the amount of inter-sites network traffic is still the same.

A last point we mention is that our approach uniformly balances the data stored on all the nodes composing the site. If a client sends a huge amount of data to a specific IPFS node, the node will have to store these data in the standard approach but in our proposal, the data is spread among all the storage servers composing the Scale-Out NAS within a site.

	IPFS	IPFS coupled with Scale-Out NAS (proposed solution)
Data locality	Yes	Yes
Network containment	No	**Only for local access**
Disconnected mode	Partially	**Yes**
Mobility support	Natively	Natively
Scalability	Yes	Yes

Table 3. Summary of Fog characteristics met by IPFS and our proposal.

3.1.3. Limitations

It is important to notice the performance of the approach we presented can greatly be improved. When an object stored remotely is accessed, the object is downloaded from all the nodes where a replica is stored (see Figure 7(c)). This is the default behaviour of IPFS to download the object in the most efficient way: if a node is overloaded, all others nodes are able to reply.

In our approach using the Scale-Out NAS, the situation is different because at most, one node per site is known in the DHT as storing a replica of the object: either the node on which the client writes the object or the first node of each site to access the object. As it is illustrated in Figure 9(c), the IPFS Node 4 located on Site 1 can only download the object from the IPFS Node 1 located on Site 1 because it is on this node the object was previously written on and therefore, it is this node which is known in the DHT as having the object. The IPFS Node 4 has no way to know that it can also download the object from the IPFS Node 2, sharing the distributed filesystem with the IPFS Node 1. This situation can lead to bottlenecks or to the overload of some nodes. In the same way, if the IPFS Node 1 becomes unreachable, the object cannot be downloaded on the other sites, whereas the IPFS Node 2 could be used as a source of the transfer.

A simple solution to these difficulties could be to store in the DHT an identifier of the site rather than an identifier of the IPFS node. We nevertheless let this modification for a future contribution.

3.2. Reduction of inter-sites network traffic for accessing data stored on a remote site

In the previous section, we proposed to couple IPFS with a Scale-Out NAS deployed independently on each site to contain the network traffic while accessing to locally stored object. However, for accessing objects stored on a remote site, the use of the DHT does not contain the network traffic close to the user and does not store the location of the object close to the replicas. In the following section, we will present the Domain Name System protocol that have interesting properties for a Fog infrastructure and then, we will introduce our own protocol relying on a tree built according to the physical topology.

3.2.1. Domain Name System protocol: an inspiration for our approach

Because neither the DHT nor any other traditional approach are 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 most able 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 benefit.

3.2.2. Metadata management overview

We propose to distribute the location records within a tree, in the same way as in the DNS protocol, the different names are spread in different servers organised in a hierarchical way. The tree is composed of different sites of Fog and contrary to the DNS, it is browsed in a bottom-up manner, from the current site to the root node. If the location is not found at a given level, the parent node is then requested.

We suppose the tree is built according to the physical topology of the network and the network latencies with the ultimate goal of reducing the time to locate any piece of data. 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.

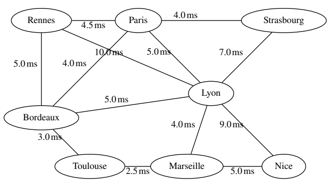


Figure 10. Part of the French National Research and Education Network **physical** topology.

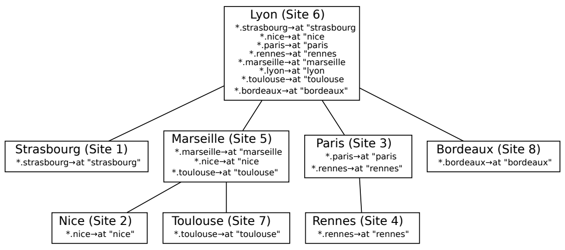


Figure 11. Tree computed with our algorithm showing the initial content of the “location servers”. Each site also has storage nodes and clients which are not represented.

Figure 11 shows the tree we computed from the network topology of the French NREN presented in Figure 10. This approach to build the tree will be explained in the next section. Figure 11 shows the location record organisation as well. The edges between the nodes correspond to physical network links. Each node is able to answer to all requests for objects stored in their subtree, and more specifically, the root node located in

Lyon is able to provide an answer to all the requests. The root node was chosen because of its central position in terms of latency according to the geographical topology (central east of France). As explained by Dabek et al., network latency is dominated by the geographic distance [DCKM04]. 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 it is not stored locally (*i.e.*, the NAS server introduced in the Section 3.1 for instance). The "location server" is responsible for storing the association between an object's name and sites in its subtree, on which a replica is stored. 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 11 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 Figures 12, 13 and 14 to explain how object replicas are created, accessed and deleted through this overlay. This figure shows the network messages exchanged by these nodes but also the physical path taken for routing them.

Object creation: Figure 12 shows when a client writes an object, a suffix corresponding to the site where the object has been created, is added to the object's name. This leads not to 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 have been 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 it does not exist a closer replica in the network. Suffixes are only used here to limit the amount of updates messages when new object replicas are created.

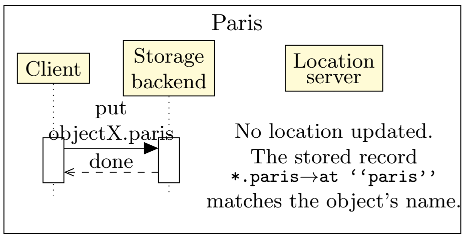
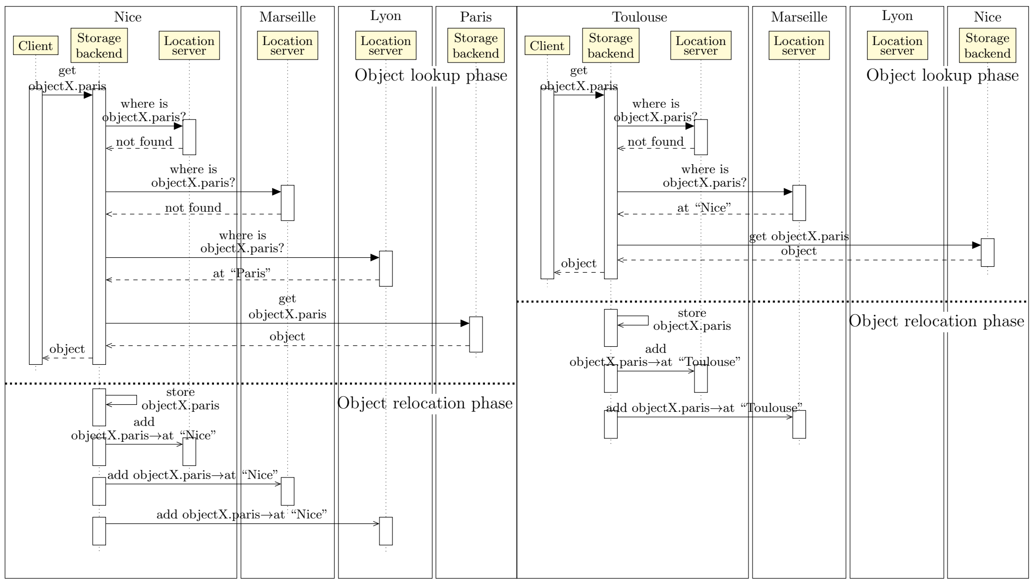


Figure 12. Sequence diagram of network traffic when a client writes an object in Paris.

Accessing an object from a remote site for the first time: Figure 13(a) shows the reading process to get 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. In this way, 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 directly replicate all object across the path between the sites and the root node. However, this strategy cannot be envisioned due to storage space it would require.

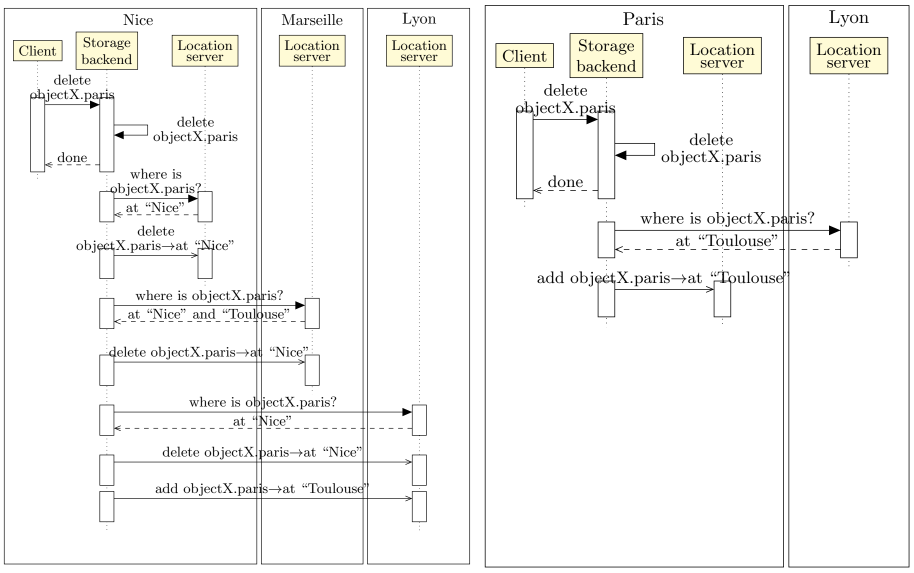
Accessing the object when several replicas are available: Figure 13(b) shows that when Toulouse requests the object created in Paris and previously relocated in Nice. The reading process is the same as previously described but the replica in Nice is accessed thanks to the location record found in Marseille. The root metadata server in Lyon is neither reached nor updated. We note that, despite a replica of the object was previously added at 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 at Nice. Therefore, suffix in object names does not have meaning in the read process. Readers only need to know the site 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. Also, in our approach, the object replica is always found in the subtree of the node we get the location of this object replica. 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).



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

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

Figure 13. Sequences diagram of network traffic when a client reads from “Nice” and “Toulouse” an object stored in “Paris”.



(a) – Delete a single object replica (Nice). (b) – Delete a "master copy" (Paris).

Figure 14. Sequences diagram of network traffic when a client deletes a single replica.

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. All the location records between pointing to this specific deleted replica are removed. But, it can lead to inconsistencies where nodes do not know another replica is stored in their subtree. That is why we propose to recopy some location records to the upper levels of the tree. For instance, to delete the replica located in Nice, the location record objectX.paris→at "Nice" is removed from the location servers of Nice and Marseille and Lyon. Nevertheless, because the location server in Marseille has also a record for this object pointing to Toulouse, a new record pointing to Toulouse has to be added in Lyon. 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 recopy 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→at "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 deleted because they are also used for other objects.

To conclude this section, we argue our protocol is more adapted for Fog infrastructures than the DHT because location is found along the physical path from the current node to the root node. Finally, in addition to reducing the lookup latency, creation of location records enables the sites to locate reachable objects replicas in case of network partitioning, increasing Fog sites autonomy. The properties of the proposed protocol are summarised in Figure 15. Our protocol limits the network traffic exchanged while locating an object (at most $log(N)$ nodes are contacted when the tree is balanced) and thus the impact on the access times. A second advantage is that it also limits the minimal number of replicas of location records needed because non replicated objects do not have a specific location record thanks to the wildcard. Finally, each node has a limited knowledge of the topology. Each node only knows its parent node in the tree. We note the value for the network traffic generated when adding or removing a site is not accurate because we did not study our protocol in such a situation.

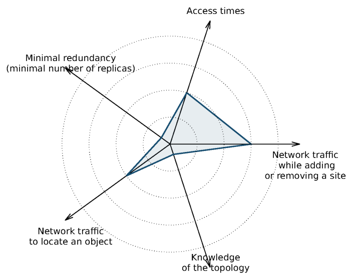
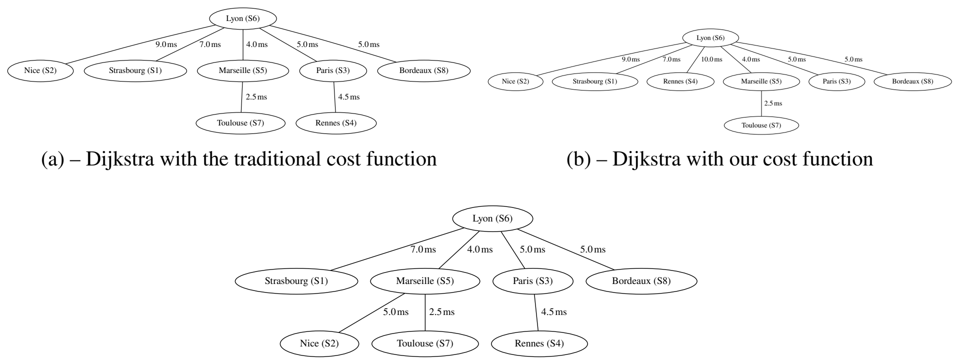


Figure 15. Star diagram summarising the characteristics of our proposed approach.

3.2.3. Algorithm to build the tree

Our protocol is relying on a tree that has to be built with care. For instance, a flat tree as shown in Figure 19(b) does not enable nodes to benefit from new location storage replicas because the root node is directly reached. Contrary to this, the other extreme is the deep tree topology as shown in Figure 19(c) where the huge number of hops leads to a poor performance. Indeed, the structure of the tree determines the maximal number of hops to locate an object but also the opportunities for other nodes to find a close object replica and the latency to locate it.



(a) – Dijkstra with the traditional cost function

(b) – Dijkstra with our cost function

(c) – Dijkstra with our relaxed cost function

Figure 16. Trees generated from the French NREN physical topology using different costs functions

The classical algorithm to compute the shortest paths from a source node to all the other nodes is the Dijkstra's algorithm [Dij59]. This list of the shortest paths from a source node to each 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 16(a), the tree has a weight of $9.0+7.0+4.0+5.0+5.0+(2.5+4.0)+(4.5+5.0)=46$. But 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 as 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 at 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.

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

The result of this modification is seen in Figure 16(b). Although this tree optimizes the total latency, it is very flat and most nodes directly access the root without benefit 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 [AHHK93] to relax the constraint. We connect a node through a specific link in the tree if the evaluated position of the node is deeper than its current one and 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 increased a little bit, a deeper node has more ancestors and a greater chance to find a location record among them. Figure 16(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 16(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.

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

To compute the value $p(j)$, we consider a uniform workload among all the sites, *i.e.*, a given object as an equal probability to be accessed from any Fog site. For instance, with the tree shown in Figure 16(b), locating an object from Marseille requires in average $0 \times \frac{1}{7} + 4.0 \times \frac{6}{7} \approx 3.43\,ms$. We consider the object replica is not located locally in Marseille but on any of the 7 others 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 the Equation 2 on the tree of the Figure 16(b). It shows an average of 45.4 ms are required to locate an object.

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

Equation 4 shows the same computation performed on the tree of the Figure 16c. It enables us to conclude that the average 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).

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

3.3. Overhead

We now evaluate the overhead of our approach compared to a DHT. First, in our approach, all the servers contacted to locate a replica are then updated once the new replica created whereas in a DHT, the number of update messages is a constant and depends on the replication level. So, in our approach the number of update messages sent can be in the worst case $O(\text{depth}(\text{tree}))$ whereas in a DHT, it is a constant $O(k)$ with k, the replication factor.

Nevertheless, in our approach, this number of update messages varies with data movements. It can be $O(\text{depth}(\text{tree}))$ for the first access of the object but it will decrease for future accesses. The more an object is accessed, the less the number of hops to locate a replica and thus the number of update messages. Therefore, if we consider an object is read successively on all the sites, in our approach the number of inter-sites update messages is equal to the number of edges in the tree ($O(n-1)$, with n the number of sites) whereas in the DHT, it is the number of sites multiplied by the replication factor of the DHT ($O(n \times k)$). In this sense, we can say that for objects accessed from a large number of sites, our approach is less costly than using a DHT.

Although, in the DHT, network traffic to maintain routing tables may be important, we should not forget to evaluate the complexity to build our tree: the Dijkstra's algorithm with a complexity of $O(n^2)$ is executed with each node as a source, therefore, the total complexity is $O(n^3)$. In this sense, strategies need to be found, to be able to deal with the network dynamicity without recomputing the whole tree each time a site is added or removed.

3.4. Resources consideration

In our protocol, we have not considered the available resources on the site a client is connected to meet their requirements. Clients always write on the closest site in terms of latency, but other metrics can be considered, such as the throughput and the available resources. This can be seen as an optimisation problem with constraints. Such a problem can be solved at a client level: the clients determine by themselves the best sites where they have to write. The main drawback of such approach is that it leads the clients to know all the network topology as well as all the sites characteristics. Another approach would be for the clients to connect the closest site and this first site determines the best location to store the objects.

We finally note that the cache replicas (*i.e.*, not the master) can be automatically deleted according to a Least Recently Used (LRU) policy in order to free the space usage corresponding to objects that were requested at some point but are not used anymore.

3.5. Conclusion and Discussion

We first presented how to reduce the inter-sites network exchanges when a locally-stored object is requested. We proposed to couple IPFS with a Scale-Out NAS, so that all the objects stored on each site are available to all the IPFS nodes of the site. Then, we proposed to reduce the inter-sites network traffic when a remotely-stored object is accessed by replacing the DHT used for managing object locations (*i.e.*, metadata management) with a protocol relying on a tree built with a modified version of the Dijkstra's algorithm taking into account the physical network topology.

Nevertheless, many improvements in our protocol are still needed. With immutable objects, the consistency is not an obstacle but enabling users to modify their data is a missing feature for an advanced object store. But, managing consistency is a real challenge because a modification requires to modify all the other replicas. This can be done asynchronously to limit the impact on access times but in this case, our protocol should be modified to become a protocol to locate a close replica at the recent version.

In its current state, our proposed Fog architecture can be seen as a symmetric or reversed CDN. Content Distribution Networks (CDN) consist in caching data on servers located close to the users. This has two advantages: *(i)* accessing to a close server can lead to reduce the access times to requested resources and *(ii)* it reduces the load on the central server because it only has to send the content to the caching server and not to every client. This technique improves the . Akamai [NSS10] or Cloudflare are well-known public CDN but companies like Google [GALM07] deploy their own CDN infrastructure to improve the QoS of their services. There is no standard architecture for these networks. Some rely on a tree and a hashing function [KLL+97,TT05] while other rely on a DHT [KRR02] and some are hybrids and rely on a mix of different protocols [EDPK09].

For several authors, the Fog hierarchy can be considered as a Content Distribution Network where object replicas are created on the fly, close to where they are needed [MKB18]. For instance, for Yang *et al.* [YZX+10], storage servers at the Edge of the network are used to cache data stored in the Cloud, which is an approach similar to what is proposed in this chapter. However, for us, the main difference between a CDN and a storage solution for Fog Computing is that in a CDN, data are stored in the core of the network and are propagated to the edge, while in the Fog, data are created at the edge. In that sense, Fog Computing can be seen as a reversed Content Distribution Network (CDN) where data sent from the clients pass through all the hierarchy until reaching the Cloud [SSX+15,BDMB+17].

Also, because our approach does not use a network overlay, it may be compared to an Information Centric Network as well. Information Centric Network (ICN) and more particularly Named Data Network (NDN) or Networking Named Content (NNC) is a concept consisting in using the network itself to directly route the requests to the node storing the requested data [JST+09]. Contrary to every other approach, there is no need to locate data before accessing them. The client just sent the request to the destination of a data and not to a specific computer and the network forwards it to the computer storing it. Several implementations have been proposed [KCC+07,JST+09,HBS+17]. Some have been specialised to work within a datacenter [KPR+12] while other are adapted to work in a multi-sites environment and especially in a CDN network [Pas12,WW17]. The main advantage of this approach is not to require a metadata server or any protocol to locate the different objects stored in the network, improving the access times. In this sense, this approach can be used to store data in a Fog Computing architecture. But, maintaining in each router a table storing the location of each object is costly.

In the next section, we will experimentally evaluate our propositions, to measure what are their impacts on the amount of network traffic exchanged between the sites but also on the access times.

4. Experimental evaluations

In this section, we discuss the evaluations we performed regarding our IPFS version for Fog environment. All the experiments are performed on a cluster of the Grid'5000 platform [BCAC+13]. The specificities of the emulated environment, such as latencies or constraints in the network connectivity, are added virtually thanks to Linux tools.

4.1. Measuring the benefit from Coupling IPFS with a Scale-Out NAS

We first evaluate our coupling of IPFS with a Scale-Out NAS.

In order to IPFS be able to store objects in a distributed filesystem, we made some modifications in the source code:

- We changed the path used to store the objects to use the Distributed filesystem without moving the local database in which each node stores its node identifier and its internal state;
- We removed the cache used in the `blockstore` module of IPFS. This cache is used to keep trace of existing objects so that nodes know they do not store the requested object and as a consequence, directly contact the global DHT. By removing it, we force the nodes to first check the presence of the object in the Distributed filesystem before contacting the DHT.

4.1.1. Material and Method

We evaluate our approach using 3 different architectures:

- IPFS in its default configuration, emulating a deployment in a Cloud Computing;
- IPFS deployed in a Fog/Edge infrastructure, without Scale-Out NAS;
- IPFS deployed in a Fog/Edge infrastructure with a local Scale-Out NAS deployed independently on each site.

We choose to use the RozoFS as Scale-Out NAS [PDÉ+14]. RozoFS is an open-source solution providing a high throughput both for sequential and random accesses. It relies on a centralised metadata server to locate the data and on an erasure code to dispatch the data between the storage nodes. The erasure code brings fault tolerance to the distributed filesystem. Mojette projections are computed for each part of the file and only 2 projections out of 3 are necessary to reconstruct the data. We emphasize this aspect because it leads to an overhead in writing (50% overhead in size *i.e.*, one redundant projection to combat one failure).

In our deployment, we consider that each storage node acts as both an IPFS node and a storage node of RozoFS (in fact, IPFS is a client of RozoFS through the `rozofsmount` daemon). To avoid any bias, we used `tmpfs` as the low level back-end for the three evaluated architectures and drop all caches after each write or get operation.

The topology we evaluated corresponds to the one illustrated in Figure 8. The platform is composed of 3 sites, each containing 6 nodes: 4 storage nodes, a metadata server for RozoFS and a client. The Cloud-based IPFS is composed of 12 IPFS nodes (the same number of nodes as used in the Fog experiments). Topology does not vary so that we do not have any node churn. The one-way network latencies between the different nodes have been set in order to be representative to:

- $L_{Fog} = 10\,\text{ms}$ is the network latency between clients and the closest site of Fog. It is the latency of wireless link [JREM14];
- $L_{Core} = 50\,\text{ms}$ is the network latency between sites of Fog. It represents the mean latency of a wide area network link [MTK06];
- $L_{Cloud} = 100\,\text{ms}$ is the network latency for a client to reach a cloud platform [SSMM14].

Network latencies are emulated thanks to the Linux Traffic Control Utility (command `tc`). Bandwidth was not modified and is set to 10 Gbps. Nevertheless, the TCP throughput is already impacted by network latency [PFTK98]. With the `iperf` tool, we measure 2.84 Gbps between clients and Fog sites, 533 Mpbs between Fog sites and 250 Mpbs to reach a Fog site.

We consider two scenarios: The first scenario of the evaluation consists in writing objects on a site and read them from the same site. The second scenario we consider consists in writing objects on a site and read them (twice) from another site. For each object written, the client connects a node on the site that is chosen randomly. We measure the time to access the object (with `YCSB` [CST+10]) but also the amount of network traffic exchanged (thanks to `iptables`) To get consistent results, the test is performed 10 times. We first evaluate the performance of our approach with local accesses before checking how it behaves in a remote reading context.

4.1.2. First scenario: writing and reading from the same site.

Table 4 shows the mean access times to write or read an object on the different infrastructures proposed. There is one client on each of the three sites and the number of objects indicated, is the number of objects read or written by each client. Therefore when 10 objects are read, it is in fact, $3 \times 10 = 30$ objects that are accessed.

Table 4(a) shows the access times for a Cloud infrastructure. In this scenario, each client writes the object on a IPFS cluster reachable in 200 ms rather than on their closest site. Results show time increases with object size: 1.72 s are needed to create a simple object of 256 KB whereas writing an object of 10 MB requires 3.07 s. The time to send the object through the high latency network links becomes important. If we increase the number of accessed objects, we observe the maximum throughput we can reach is almost 290 Mbps ($100 \times 10/27.58 = 36$ MBytes per second). For small objects, the situation is even worse 11.18 Mbps ($0.256 \times 100/2.29$). Nevertheless, access times are better with small objects for several reasons. First, the DHT is not used when the node storing an object is the node the object was written on. And secondly, the TCP throughput is better from the IPFS node to the client than the contrary. This is due to the management of TCP

Mean **writing** time (seconds)				Mean **reading** time (seconds)			
Size / Number	256 KB	1 MB	10 MB	Size / Number	256 KB	1 MB	10 MB
1	1.72	2.14	3.07	1	1.47	1.88	3.04
10	1.53	2.00	7.97	10	1.35	1.77	5.22
100	2.29	5.55	27.58	100	1.57	2.62	11.24

(a) – By using a centralized Cloud to store the objects.

Mean **writing** time (seconds)				Mean **reading** time (seconds)			
Size / Number	256 KB	1 MB	10 MB	Size / Number	256 KB	1 MB	10 MB
1	0.17	0.22	0.34	1	0.25	0.28	0.54
10	0.17	0.21	0.40	10	0.26	0.27	0.54
100	0.33	1.07	3.92	100	0.29	0.50	1.98

(b) – By using the default approach of IPFS.

Mean **writing** time (seconds)				Mean **reading** time (seconds)			
Size / Number	256 KB	1 MB	10 MB	Size / Number	256 KB	1 MB	10 MB
1	0.18	0.23	0.38	1	0.14	0.18	0.31
10	0.17	0.22	0.43	10	0.14	0.18	0.36
100	0.33	1.08	3.97	100	0.19	0.36	1.83

(c) – By using IPFS on top of RozoFS distributed filesystem, deployed independently on each site.

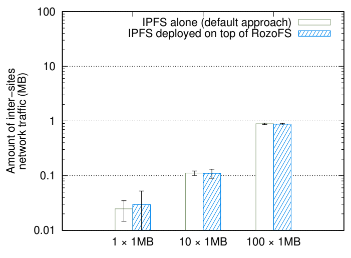
Table 4. Mean times (in seconds) to write and read an object by a client on the local site.

connections. The IPFS client sends data in small packets whereas the server try to send them in bigger ones.

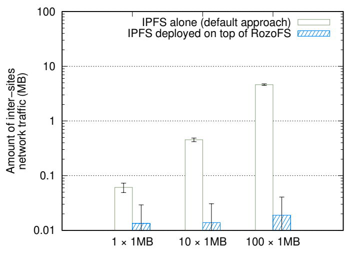
Table 4(b) shows the access times measured when IPFS is deployed in a Fog Computing environment. The results clearly shows the interest of deploying a storage solution at the Edge of the network. Access times are clearly better than there were previously with a maximum throughput of 4 Gbps (500 MBps), which is half of the maximum capacity of the 10 Gbps link. Secondly, performance are reduced due to high latency network links composing the DHT, which was not the case in a Cloud deployment, (0.5 ms latency between the nodes).

Table 4(c) shows the access times when IPFS is deployed on top of RozoFS. First of all, we observe that in the two deployments in a Fog environment, writing times are similar. 3.92 seconds are needed per objects to write 100 objects of 10 MB on each site when IPFS is deployed alone and 3.97 s when it is coupled with RozoFS. In other words, adding a Scale-Out NAS does not change the performance. This can be surprising because we can think that adding a layer can add an overhead.

In reading, using RozoFS leads to improve the access times by 34 %. This is especially true for small objects, because for these objects the time to access the DHT is important in regard to the total access times. For instance, it takes 0.14 s to read 10 objects of 256 KB when IPFS is on top of RozoFS and 0.25 s when IPFS is deployed alone (almost the double of time). We also note the clients choose randomly the IPFS node they read the object from. Therefore in Table 4(b), there is $\frac{1}{4}$ of chance the client contacts the node the object was previously written on. Leading on to use the DHT. This probability will decrease as long as Fog Sites contains more servers, leading to a general



(a) – Write



(b) – Read

Figure 17. Amount of inter-sites network traffic exchanged between the sites while clients write and read objects on their local site.

Figures 17(a) and 17(b) show the quantity of network traffic exchanged between the sites while clients are writing and reading objects on their local site. We do not present the results when IPFS is deployed in a Cloud architecture because in this case, the exchanged traffic between sites corresponds to the amount of data sends to or reads from the Cloud.

We observe the amount of traffic exchanged between the sites only depends on the number of accessed objects but not on their size. For a better visibility, we present the results for objects of 1 MB. Figure 17(a) shows the amount of network traffic exchanged during the writing. As expected, this quantity is equivalent in the two approaches because the DHT is updated asynchronously each time a new object is written.

For the reading, the default approach shows the more manipulated objects, the more the amount of network traffic. In our approach using RozoFS, we only observe a small amount of traffic sent regularly to maintain the routing table of the DHT (13 KB for the presented case). Therefore, the result depends only on the time taken to realise the experimentation.

To conclude on this experiment, adding a Scale-Out NAS does not impact the performance to write. Writing times are similar both when IPFS is used alone and when it is coupled. In reading, using a Scale-Out NAS avoids using the DHT when the requested objects are stored on the local site. This improves the access times, especially for small objects for which accessing the DHT is longer than transferring the object data. At the same time, not using this DHT leads to reduce significantly the inter-sites network traffic.

4.1.3. Second scenario: writing on a site and reading from another site.

We now evaluate how our approach when accessed objects are not stored on the local site. A first client writes an object on the local site and then, another client, located on another site, read the object that has just been written. We only present reading access times because writing ones are similar to what we presented in the previous analysis.

Table 5(a) and 5(b) show the reading times for two successive reading. For the first read, the two approaches get a similar access time. Indeed, for the two approaches, the requested object is not stored on the local site. The DHT needs to be accessed to locate the object and then a replica has to be downloaded. This corresponds to what we described in Figures 7(c) and 9(c).

For the second read, we however observe better access times with our approach. For instance, with RozoFS, when 100 objects of 10 MB are read in parallel, 1.86 s per objects are required whereas it takes 6,08 s without RozoFS. This huge difference can be

Mean reading time (seconds) **First read**				Mean reading time (seconds) **Second read**			
Size / Number	256 KB	1 MB	10 MB	Size / Number	256 KB	1 MB	10 MB
1	1,39	1,92	13,07	1	1,01	1,85	3,70
10	1,01	1,92	6,48	10	0,70	1,31	5,95
100	0,94	2,02	9,76	100	0,71	1,37	6,08

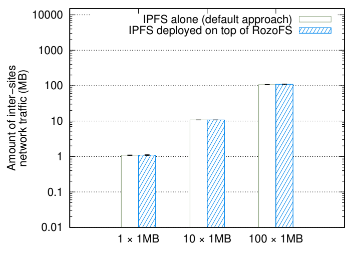
(a) – Using IPFS alone.

Mean reading time (seconds) **First read**				Mean reading time (seconds) **Second read**			
Size / Number	256 KB	1 MB	10 MB	Size / Number	256 KB	1 MB	10 MB
1	1,35	3,86	13,21	1	0,15	0,19	0,31
10	1,11	2,17	8,40	10	0,14	0,19	0,35
100	1,09	2,51	9,22	100	0,33	0,46	1,86

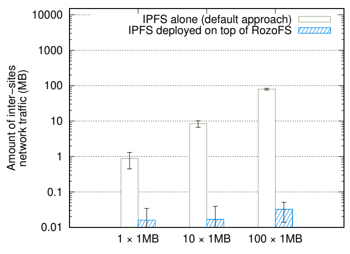
(b) – Using IPFS deployed on top of a RozoFS cluster deployed independently on each site.

Table 5. Mean times (in seconds) to read an object twice with IPFS, using the default approach (a) and our coupling (b).

explained because in our approach, the object is downloaded from the local Scale-Out NAS but in the default approach the DHT is accessed and then, two object replicas are found: the first one and the replica created on the local site during the previous access. But IPFS does not know a replica is reachable with a low latency and the other one is reachable through the high latency links.



(a) – First read



(b) – Second read

Figure 18. Amount of inter-sites network traffic exchanged between the sites while clients read objects stored remotely.

Figures 18(a) and 18(b) show the amount of network traffic exchanged between the sites for the two succesive reads when the data is not located on the local site. For the first read, the same amount of network traffic is exchanged in the two approaches: as explained previously, an object replica has to be located using the global DHT and downloaded from the remote site. For the second read, we observe a drastic reduction of the network traffic. In the two approaches, two replicas of each object are available: the remote one, and the local replica that was created in the first read. Nevertheless, in the default approach, IPFS is not able to favorise the local replica. The DHT is accessed to locate the two object replicas before downloading them in parallel in order to speed up the process. This explains why the remote replica is accessed and the inter-sites network traffic is not reduced. In our approach, the nodes the requests are sent to retrieve the objects from the local Scale-Out NAS, avoiding all the exchanges between the sites.

4.1.4. Conclusion

The main results of this evaluation are:

- Scale-Out NAS does not impact the performance in writing, with an interesting side effect as 50% of extra data are stored for fault tolerance;
- Local reads are satisfied with the underlying distributed filesystem, avoiding using the DHT and increasing the performance by 34%;
- The same amelioration is observed for remote reads, once a new replica is created on the local site.

4.2. Evaluation of our location protocol relying on a tree

We now evaluate the second proposition we made. To understand how the location protocol relying on a tree behaves, we first evaluate it on micro benchmark before performing an evaluation on a real topology.

4.2.1. Material and Method

In this evaluation, each site is composed of only one IPFS node that also acts as a "location server". To mitigate our development effort, "location servers" are implemented in DNS servers deployed independently on each node. These DNS are used as key/value store to save the location records. The DNS servers provide us 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 messages.

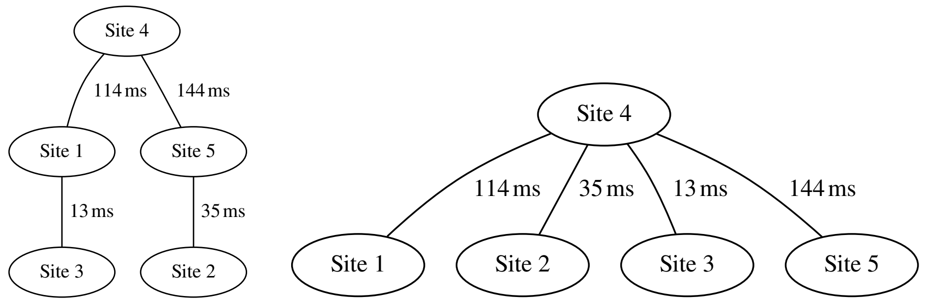
More specifically, we use BIND servers, configured as authoritative servers to store the records in flat text files (BIND default backend). We validated in an experiment the response time increased only after approximately 30 000 records stored on a single server. Because we store much fewer records, we argue that this backend does not impact our results. Nevertheless, we precise that BIND servers are deployed independently to enable 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[2] and in the standard one that uses a DHT[3]. The scenario consists in reading objects from several 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

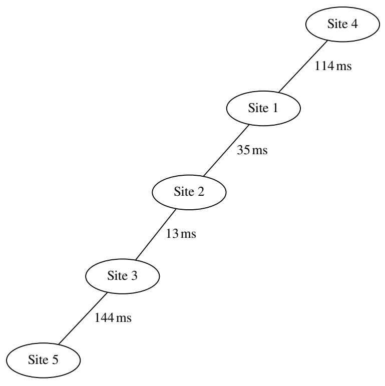
[2] https://github.com/bconfais/go-ipfs/tree/dns

[3] https://github.com/bconfais/go-ipfs/tree/dht_name_based

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. Because we only measure the time to locate objects, we note the size of objects has no impact on our results. We performed micro benchmarks using topology in Figure 19 to easily understand how our protocol behaves on simple topologies. These topologies are manually built from 5 sites of the wondernetwork matrix of latencies[4].



(a) – Balanced tree. (b) – Flat tree.



(c) – Deep tree.

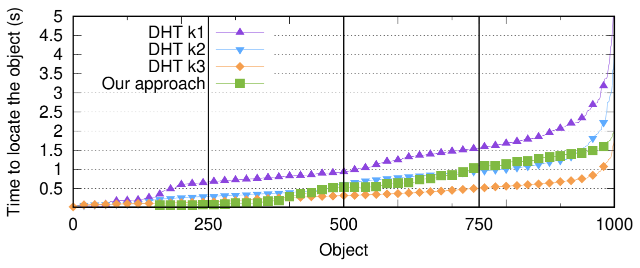
Figure 19. Topologies used for micro-benchmarks.

4.2.2. *First topology, a balanced tree*

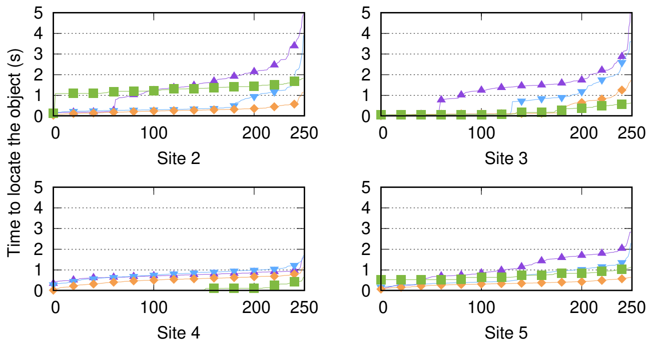
We first evaluate our approach by performing the scenario with the tree given in Figure 19(a).

The time to locate objects is shown in Figure 20(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 in the worst case 1.982 s which is faster than a DHT with only one replica (about 4.794 s) but longer than

[4]https://wondernetwork.com



(a) – All sites



(b) – Per site

Figure 20. 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 their time to determine their location.

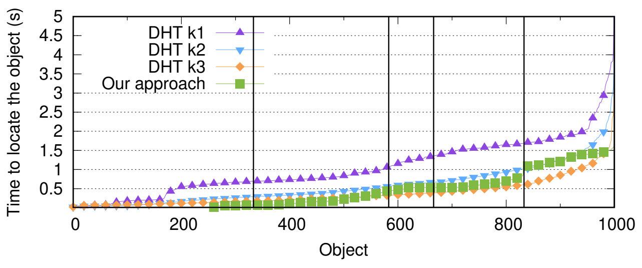
a DHT with 3 replicas (about 1.740 s). We nevertheless note that the comparison with "DHT k3" is not so totally fair in a frst read because for each object, three location records are available in the DHT whereas only one is available in our approach. Because objects are accessed in parallel, the time to locate the last object is also the time to locate the 1000 objects. 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 20(a) and next ones 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). Objects from 500 to 750 are read from Site 5. Finally, after the last threshold, we observe the objects read from Site 2 requiring a first hop to Site 5 and a second hop to Site 4 to locate them. The non-linearity we observe close to those lines means the observed result is what we expect. Because objects are sorted, theoretical thresholds do not delimit exactly what is happening site by site. To deeply understand their individual behaviours, we split the Figure 20(a) according to the site requesting each object.

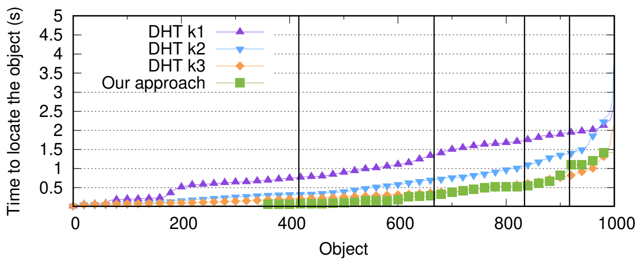
Figure 20(b) shows the time to locate an object is not the same for each site because the network latency to connect them is different. For instance, in the DHT, Site 4 cannot reach another node in less than 114 ms because it does not have closer neighbour whereas

Site 3 can reach Site 1 in 13 ms. We do not observe non-linearity in our approach for a given site because each of them finds all the location records from the same location.

We note in Figure 20(a), the tail from object 920 to 1000 is due to a bad parallelism of IPFS we checked with sequential accesses. 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 bias, we next evaluate the performance in terms of hops rather than in absolute time.



(a) – Second read



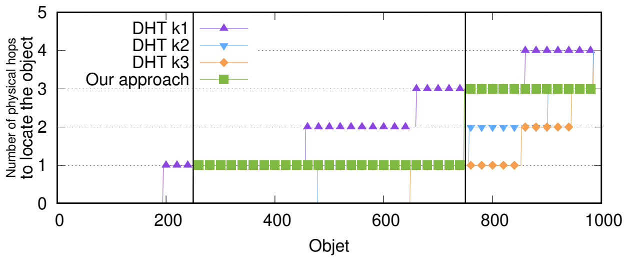
(b) – Third read

Figure 21. Times to find the location of objects in the second read for all sites (a) and for the third read (b).

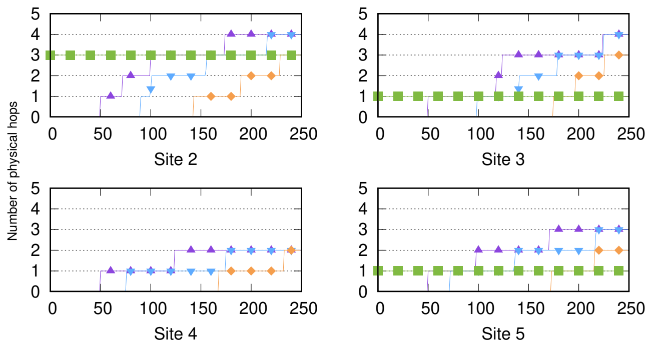
Figure 21 shows in further reads, the time to locate the objects does not vary with the DHT but decreases in our approach that creates new location records when objects are accessed. For the third read, our approach becomes better than the approach using 3 replicas in the DHT: location records are close 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 21(a), location is found locally for 333 objects because Site 5 now store the location for objects that have been read from Site 2 during the first read. A similar observation is made on Site 2 for which objects are located from Site 5 instead of Site 4.

Figure 22 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 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 2 physical links). The result is similar to



(a) – All sites



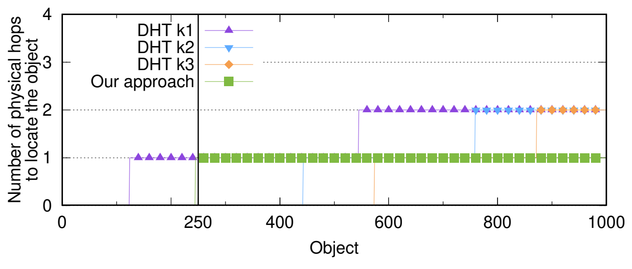
(b) – Per site

Figure 22. 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.

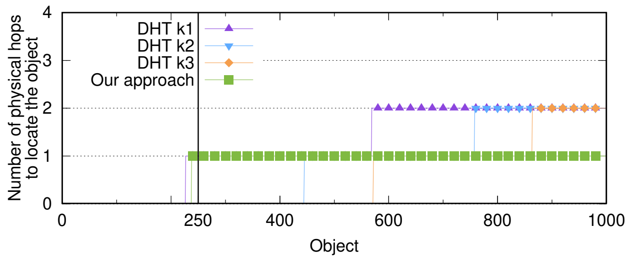
what we observed in Figure 20(a). But contrary to this figure, the number of physical hops are not impacted by the network latencies. Therefore, implementation biases that could speed up or slow down location times do not impact this result. The conclusion of this experiment is that by requesting close nodes first and by creating new location records read after read, our approach provides better performance than the DHT.

4.2.3. Flat tree topology

The second topology we evaluate is shown in Figure 19(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. The reason is that network latencies are taken into account when the tree is built so that the network latency to reach the root node is the lowest as possible. We focus on the number of hops in order not to consider the different possible latencies when a leaf node reaches another leaf node in the DHT. Figure 23 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 determined by reaching Site 4. The theoretical thresholds delimit 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. The conclusion on 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 and these hops have the lowest latency possible due to the algorithm used to build the tree. We however



(a) – First read



(b) – Fourth read

Figure 23. 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.

note that a more important number of objects or peers, may overload the root node. This highlights the importance of having a well-balanced tree.

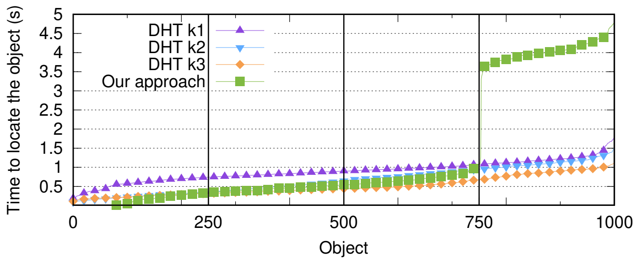
4.2.4. Deep tree topology

The last topology we evaluate is the other extreme case in which all the sites are organised in a very vertical tree as shown in Figure 19(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 24(a) that reading the 1000 object in 4.5 s in our approach makes it worse than the DHT which only needs 1.7 s. Figure 24(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. 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.

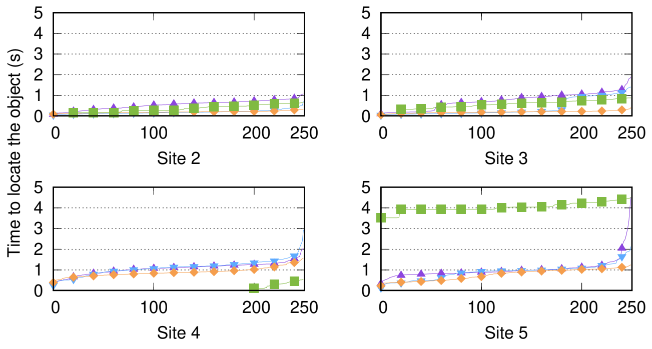
4.2.5. French NREN, a real topology

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

We consider the graph of a part of the French NREN network shown in Figure 10. In order to evaluate our approach, we use the tree in Figure 11 that have been computed



(a) – All sites



(b) – Per site

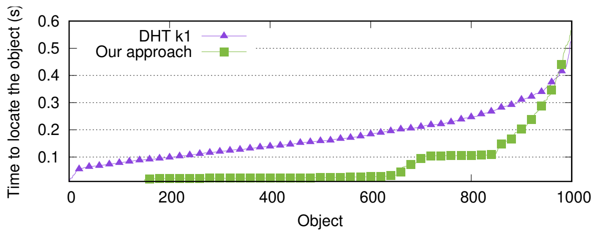
Figure 24. Times to find the location of objects in the first read for all sites (a) and for each site (b) for the deep tree topology. Objects are sorted by their time to locate them.

using our approach presented in Section 3.2.3. 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.

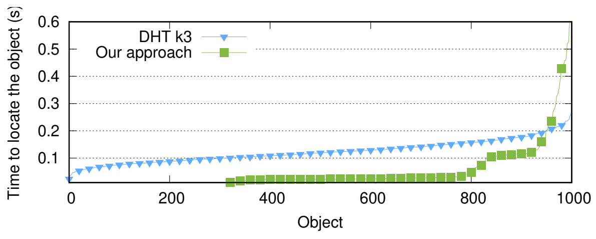
We perform the same experiment as in the previous section. 1000 objects are written on Site 1 (Strasbourg) and are read successively from other sites.

Figure 25 shows the times to access the location records in the first read, the third and in the seventh read. Because of the high number of sites, we performed the experiment with the DHT up to 6 replicas for a fair comparison. We observe that for all reads, our approach has a better performance than the DHT, especially because in our approach, the closest nodes are requested first. In Figure 25(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).

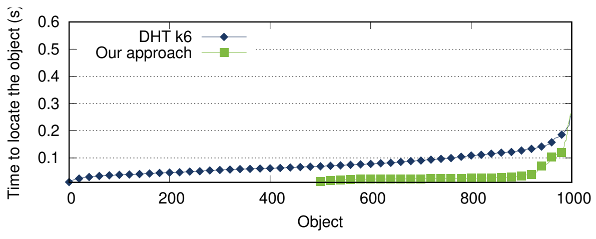
In the third read, shown in Figure 25(b), our approach becomes better than the DHT with 3 replicas because of the relocation even if 5% of objects (the last 50 objects) are read with a longer access time because these objects have not benefited from the relocation. For instance an object read in Lyon, 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 25(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

Figure 25. Times to locate objects in the first, the third and the sixth read.

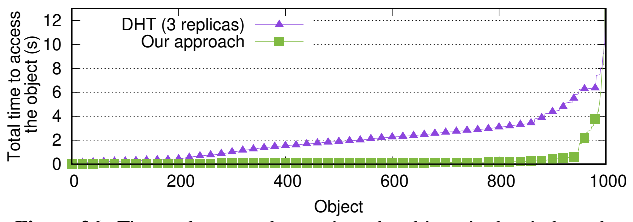


Figure 26. Time to locate and to retrieve the objects in the sixth read.

Figure 26 shows the total amount of time to locate the objects and to download them on the site 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 only the curve for 3 replicas.

4.2.6. Conclusion

These experiments show that by limiting the amount of hops, 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 benefit from these characteristics.

5. Coupling Grid’5000 and Fit/IoT-Lab testbeds: a platform for Fog experimentations

After evaluating our approach on the Grid’5000 platform, where resources are abondants and are not a limiting factor, we want to evaluate how our protocol behaves in a more realistic environment, where the Internet of Things is interconnected to the Internet of Servers. To the best of our knowledge, such a platform to test Fog applications does not exist. In this section, we propose a specific environment coupling two testbeds: the Grid’5000 and the FIT/IoT-Lab platforms. The FIT/IoT-Lab platform provides connected sensors [ABF+15]. Using this platform in conjunction with Grid’5000 can enable us to evaluate our approach with real sensors producing real data with a real workload. We precise we found another work in the literature focusing on the interconnection of these two platforms [DFLP18].

5.1. FIT/IoT-Lab platform

FIT/Iot-Lab is a testbed providing different kind of sensors with few computing resources and few memory. “M3 nodes” use an ARM processor and a IEEE 802.15.4 (Zigbee) radio link, limited to 250 Kbps. The nodes can be connected only with the radio link but can also act as border routers, between the radio network and the Ethernet one. Different operating systems are available such as FreeRTOS [Bar10], Contiki [DSF+11] or RIOT [BHG+13] and some nodes can be embedded in mobile robots.

A second type of nodes called “A8 nodes” are more powerful and provide a Linux environment. Figure 27 shows a general overview of the platform and like with Grid’5000, nodes are spread on different sites in France (Grenoble, Lille, Lyon, Saclay and Strasbourg). We note all the nodes are connected to an IoT-Lab gateway that can flash the ROM of the nodes and that can make the serial console remotely available.

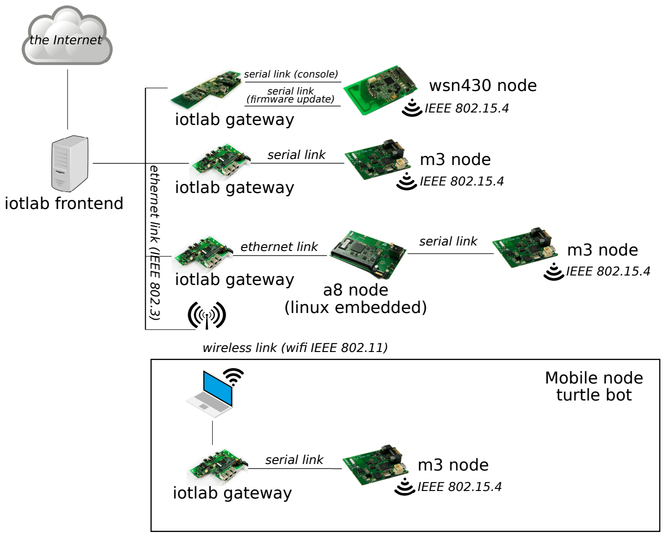


Figure 27. Network topology of a FIT/IoT-Lab site.

5.2. Mapping a Fog environment to the testbed

While the Grid'5000 platform provides large computing and storage resources and can be used to emulate a Fog site, the FIT/IoT-Lab provides sensors with low computing resources that can be used as clients of the Fog site.

In order to provide some locality, we propose to perform the experimentation on the physical locations hosting both a Grid'5000 and a FIT/IoT-Lab cluster. This is the case for the cities of Grenoble, Lyon and Lille. We can expect a direct network routing between the two platforms because they both rely on the French NREN.

5.3. Network interconnection of the two platforms

Although the two platforms are connected to the same network provider (Renater), they are not easily reachable because each of them use an IPv4 network that is not globally routed. Each platform uses a Network Address Translation (NAT) or a frontend between the nodes and the Internet making difficult to reach the FIT/IoT-Lab platform from the Grid'5000 one (and vice-versa). Also, the IPv6 protocol is only available on the FIT platform but not on Grid'5000 making impossible an interconnection using this protocol.

From the Grid'5000 platform, only the frontend of the FIT/IoT-Lab platform can be directly reached. We therefore propose to establish a SSH tunnel between this frontend and a machine located in a Grid'5000 cluster. Unfortunately the frontend can reach the nodes on the platform but it is impossible to make a port listening on the frontend to use the tunnel from the nodes of the site. This is the reason we established a second tunnel between the frontend and a A8 node. This network interconnection is presented Figure 28 and the scripts to run this experiment are available on github[5].

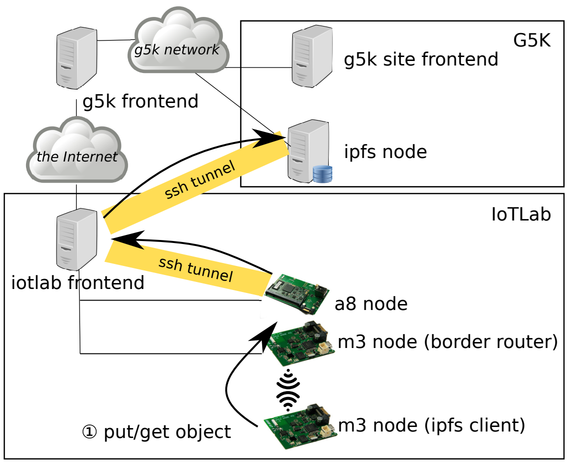


Figure 28. Double SSH tunnel to connect the Grid'5000 and the FIT/IoT-Lab platform.

Nevertheless, using a tunnel adds a significant overhead that can impact the total latency and thus the access times. Also, although the two testbeds are connected to the same network provider and are located in the same city, the routing is not optimal. All packets pass through a Grid'5000 gateway located either at Rennes or at Sophia.

[5]`https://github.com/bconfais/benchmark/tree/master/iot-lab`

Contribution	Future work
Coupling IPFS to a local Scale-Out NAS	• Solving the bottleneck issue in the DHT.
Tree protocol to locate data	• Supporting network topology changes.
General open issues	• Taking other metrics (storage capacity, workload, ...) into account rather than always accessing the closest site; • Making objects mutable and managing the consistency.

Table 6. Future work and open issues.

In conclusion, having a platform to evaluate Fog applications is a real need but coupling two testbeds like we proposed is not a straightforward solution and additional work needs to be done to make it usable for real experimentations.

Beyond this coupling, we also note that proposing a full software stack for real life Fog platforms is something required. We note that the DISCOVERY project aims to adapt the OpenStack components to such a Fog environment [LPTDC15]. We also note that few industrial Fog Computing platforms have already been deployed such as *Amazon Lambda@Edge*[6] or Akamai Cloudlet [PSW+15] even if the details of these infrastructures such as the location of the sites are not publicly available.

6. Conclusion and Future work

In this chapter, we first introduced why traditional distributed Filesystems cannot work in a multisites environment. We presented the list of characteristics we expect for a storage solution to enable IoT devices to efficiently store the data they produce before explaining why we chose the IPFS solution as a starting point to create a storage solution that can handle the massive amount of data produced by IoT devices.

We proposed to couple IPFS with a distributed filesystem deployed independently on each site so that all the nodes of a given site directly share what is stored on the site. This reduces the load on the global DHT used to locate data and thus, the access times (300 ms to 100 ms) because it is now only used when the requested object cannot be found locally. This performance improvment is very important to enable the storage solution to deal with the huge workload of the Internet of Things. Nevertheless, while this approach contains the network traffic for locally stored objects, it may unbalance the DHT in case of remote access because the DHT only knows at most one object replica per site. To reduce the network traffic generated by the global DHT, we then presented a new protocol relying on a tree mapping the physical topology. We experimentally showed that by taking into account the physical topology and by replicating the location records close to where objects are requested, our new approach contains the network traffic and reduces significantly the time to locate data. Finally, we proposed to couple two different testbeds to create an entire platform providing sensors and servers to evaluate Fog applications. Nevertheless, the interconnection of the two testbeds is not easy due to latency constraints.

As presented in Table 6, we plan to improve our coupling of IPFS with a Scale-Out NAS that currenlty leads to a bottleneck issue because the DHT reflects that only one node pers site stores each object. We also plan to investigate how we can improve

[6]`https://docs.aws.amazon.com/lambda/latest/dg/lambda-edge.html`

our tree location protocol to support dynamic network topology, when latencies between the sites change or when a site of Fog is dynamically added or shut down. This future work leads to several problematics such as dynamically recomputing the tree but also dynamically moving the location records to reflect the new tree. Enabling user to use mutable objects and managing the consistency within the different object replicas is also a challenge we also need to overcome. Finally, taking into account the resources (storage space, computation resources) available on each site rather than always connecting the closest one would be a great improvement.

References

[ABF+15] C. Adjih, E. Baccelli, E. Fleury, G. Harter, N. Mitton, T. Noel, R. Pissard-Gibollet, F. Saint-Marcel, G. Schreiner, J. Vandaele, and T. Watteyne. FIT IoT-Lab: A large scale open experimental IoT testbed. In *2015 IEEE 2nd World Forum on Internet of Things (WF-IoT)*, pages 459–464, December 2015.

[AHHK93] 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*, pages 1869–1872 vol.3, May 1993.

[Bar10] R. Barry. *Using the FreeRTOS Real Time Kernel: A Practical Guide*. Real Time Engineers Limited, 2010.

[BCAC+13] Daniel Balouek, Alexandra Carpen Amarie, Ghislain Charrier, Frédéric Desprez, Emmanuel Jeannot, Emmanuel Jeanvoine, Adrien Lebre, David Margery, Nicolas Niclausse, Lucas Nussbaum, Olivier Richard, Christian Pérez, Flavien Quesnel, Cyril Rohr, and Luc Sarzyniec. Adding Virtualization Capabilities to the Grid'5000 Testbed. In IvanI. Ivanov, Marten Sinderen, Frank Leymann, and Tony Shan, editors, *Cloud Computing and Services Science*, volume 367 of *Communications in Computer and Information Science*, pages 3–20. Springer International Publishing, 2013.

[BDL+17] 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*, 14(4):1143–1156, Dec 2017.

[BDMB+17] L. F. Bittencourt, J. Diaz-Montes, R. Buyya, O. F. Rana, and M. Parashar. Mobility-aware application scheduling in fog computing. *IEEE Cloud Computing*, 4(2):26–35, March 2017.

[Ben14] Juan Benet. IPFS - Content Addressed, Versioned, P2P File System. Technical report, Protocol Labs, Inc., 2014.

[BHG+13] E. Baccelli, O. Hahm, M. Gunes, M. Wahlisch, and T. C. Schmidt. Riot os: Towards an os for the internet of things. In *2013 IEEE Conference on Computer Communications Workshops (INFOCOM WKSHPS)*, pages 79–80, April 2013.

[BMNZ14] Flavio Bonomi, Rodolfo Milito, Preethi Natarajan, and Jiang Zhu. *Fog Computing: A Platform for Internet of Things and Analytics*, pages 169–186. Springer International Publishing, 2014.

[BMZA12] Flavio Bonomi, Rodolfo Milito, Jiang Zhu, and Sateesh 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*, MCC '12, pages 13–16, New York, NY, USA, 2012. ACM.

[Bre10] Eric Brewer. A certain freedom: Thoughts on the cap theorem. In *Proceedings of the 29th ACM SIGACT-SIGOPS Symposium on Principles of Distributed Computing*, PODC '10, pages 335–335, New York, NY, USA, 2010. ACM.

[BW15] Charles C. Byers and Patrick Wetterwald. Fog computing distributing data and intelligence for resiliency and scale necessary for iot: The internet of things (ubiquity symposium). *Ubiquity*, 2015(November):4:1–4:12, November 2015.

[CDM+12] Jianjun Chen, Chris Douglas, Michi Mutsuzaki, Patrick Quaid, Raghu Ramakrishnan, Sriram Rao, and Russell Sears. Walnut: A unified cloud object store. In *Proceedings of the 2012 ACM SIGMOD International Conference on Management of Data*, SIGMOD '12, pages 743–754, New York, NY, USA, 2012. ACM.

[CLP16] Bastien Confais, Adrien Lebre, and Benoît Parrein. Performance Analysis of Object Store Systems in a Fog/Edge Computing Infrastructures. In *IEEE CloudCom*, Luxembourg, Luxembourg, December 2016.

[CLRT00] Philip H. Carns, Walter B. Ligon, III, Robert B. Ross, and Rajeev Thakur. Pvfs: A parallel file system for linux clusters. In *Proceedings of the 4th Annual Linux Showcase & Conference - Volume 4*, ALS'00, pages 28–28, Berkeley, CA, USA, 2000. USENIX Association.

[CRS$^+$08] Brian F. Cooper, Raghu Ramakrishnan, Utkarsh Srivastava, Adam Silberstein, Philip Bohannon, Hans-Arno Jacobsen, Nick Puz, Daniel Weaver, and Ramana Yerneni. Pnuts: Yahoo!'s hosted data serving platform. *Proc. VLDB Endow.*, 1(2):1277–1288, August 2008.

[CST$^+$10] Brian F. Cooper, Adam Silberstein, Erwin Tam, Raghu Ramakrishnan, and Russell Sears. Benchmarking Cloud Serving Systems with YCSB. In *Proceedings of the 1st ACM Symposium on Cloud Computing*, SoCC '10, pages 143–154, New York, NY, USA, 2010. ACM.

[DCKM04] Frank Dabek, Russ Cox, Frans Kaashoek, and Robert Morris. Vivaldi: A decentralized network coordinate system. *SIGCOMM Comput. Commun. Rev.*, 34(4):15–26, August 2004.

[DFLP18] Bruno Donassolo, Ilhem Fajjari, Arnaud Legrand, and Mertikopoulos Panayotis. FogIoT Orchestrator: an Orchestration System for IoT Applications in Fog Environment. In *1st Grid'5000-FIT school*, Nice, France, April 2018.

[DHH$^+$03] S Donovan, G Huizenga, AJ Hutton, CC Ross, MK Petersen, and P Schwan. Lustre: Building a file system for 1000-node clusters. In *Proceedings of the Linux Symposium*, 2003.

[DHJ$^+$07] Giuseppe DeCandia, Deniz Hastorun, Madan Jampani, Gunavardhan Kakulapati, Avinash Lakshman, Alex Pilchin, Swaminathan Sivasubramanian, Peter Vosshall, and Werner Vogels. Dynamo: Amazon's highly available key-value store. *SIGOPS Oper. Syst. Rev.*, 41(6):205–220, October 2007.

[Dij59] E. W. Dijkstra. A note on two problems in connexion with graphs. *Numer. Math.*, 1(1):269–271, December 1959.

[DO13] Alex Davies and Alessandro Orsaria. Scale out with glusterfs. *Linux J.*, 2013, November 2013.

[DSF$^+$11] Adam Dunkels, Oliver Schmidt, Niclas Finne, Joakim Eriksson, Fredrik Österlind, Nicolas Tsiftes, and Mathilde Durvy. The contiki os: The operating system for the internet of things. Technical report, Thingsquare AB, 2011.

[DYC$^+$15] Harishchandra Dubey, Jing Yang, Nick Constant, Amir Mohammad Amiri, Qing Yang, and Kunal Makodiya. Fog data: Enhancing telehealth big data through fog computing. In *Proceedings of the ASE BigData ; SocialInformatics 2015*, ASE BD;SI '15, pages 14:1–14:6, New York, NY, USA, 2015. ACM.

[EDPK09] Manal El Dick, Esther Pacitti, and Bettina Kemme. Flower-cdn: A hybrid p2p overlay for efficient query processing in cdn. In *Proceedings of the 12th International Conference on Extending Database Technology: Advances in Database Technology*, EDBT '09, pages 427–438, New York, NY, USA, 2009. ACM.

[FGH14] Mohamed Firdhous, Osman Ghazali, and Suhaidi Hassan. Fog computing: Will it be the future of cloud computing? In *Third International Conference on Informatics & Applications, Kuala Terengganu, Malaysia*, pages 8–15, 2014.

[GALM07] Phillipa Gill, Martin Arlitt, Zongpeng Li, and Anirban Mahanti. Youtube traffic characterization: A view from the edge. In *Proceedings of the 7th ACM SIGCOMM Conference on Internet Measurement*, IMC '07, pages 15–28, New York, NY, USA, 2007. ACM.

[GL02] Seth Gilbert and Nancy Lynch. Brewer's conjecture and the feasibility of consistent, available, partition-tolerant web services. *SIGACT News*, 33(2):51–59, June 2002.

[HBS$^+$17] Oliver Hahm, Emmanuel Baccelli, Thomas C. Schmidt, Matthias Wählisch, Cédric Adjih, and Laurent Massoulié. Low-power Internet of Things with NDN & Cooperative Caching. In *ACM ICN 2017 - 4th ACM Conference on Information-Centric Networking*, Berlin, Germany, September 2017.

[HCK$^+$08] Felix Hupfeld, Toni Cortes, Björn Kolbeck, Jan Stender, Erich Focht, Matthias Hess, Jesus Malo, Jonathan Marti, and Eugenio Cesario. The xtreemfs architecture; a case for object-based file systems in grids. *Concurr. Comput. : Pract. Exper.*, 20(17):2049–2060, December 2008.

[JREM14] N. T. K. Jorgensen, I. Rodriguez, J. Elling, and P. Mogensen. 3G Femto or 802.11g WiFi: Which Is the Best Indoor Data Solution Today? In *2014 IEEE 80th Vehicular Technology Conference (VTC2014-Fall)*, pages 1–5, Sept 2014.

[JST$^+$09] Van Jacobson, Diana K. Smetters, James D. Thornton, Michael F. Plass, Nicholas H. Briggs, and Rebecca L. Braynard. Networking named content. In *Proceedings of the 5th International Conference on Emerging Networking Experiments and Technologies*, CoNEXT '09, pages 1–12, New York, NY, USA, 2009. ACM.

[KCC+07] Teemu Koponen, Mohit Chawla, Byung-Gon Chun, Andrey Ermolinskiy, Kye Hyun Kim, Scott Shenker, and Ion Stoica. A data-oriented (and beyond) network architecture. *SIGCOMM Comput. Commun. Rev.*, 37(4):181–192, August 2007.

[KLL+97] David Karger, Eric Lehman, Tom Leighton, Rina Panigrahy, Matthew Levine, and Daniel 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*, STOC '97, pages 654–663, New York, NY, USA, 1997. ACM.

[KPR+12] Bong Jun Ko, Vasileios Pappas, Ramya Raghavendra, Yang Song, Raheleh B. Dilmaghani, Kang-won Lee, and Dinesh Verma. Architecture for data center networks an information-centric architecture for data center networks. In *Proceedings of the Second Edition of the ICN Workshop on Information-centric Networking*, ICN '12, pages 79–84. ACM, 2012.

[KRR02] Jussi Kangasharju, James Roberts, and Keith W. Ross. Object replication strategies in content distribution networks. *Computer Communications*, 25(4):376 – 383, 2002.

[LBB14] Adrien Lebre and Gustavo Bervian Brand. GBFS: Efficient Data-Sharing on Hybrid Platforms. Towards adding WAN-Wide elasticity to DFSes. In *WPBA Workshop in Proceedings of 26th International Symposium on Computer Architecture and High Performance Computing*, WPBA Workshop in Proceedings of 26th International Symposium on Computer Architecture and High Performance Computing, Paris, France, October 2014. IEEE.

[LM10] Avinash Lakshman and Prashant Malik. Cassandra: A decentralized structured storage system. *SIGOPS Oper. Syst. Rev.*, 44(2):35–40, April 2010.

[LPTDC15] Adrien Lebre, Jonathan Pastor, and . The DISCOVERY Consortium. The DISCOVERY Initiative - Overcoming Major Limitations of Traditional Server-Centric Clouds by Operating Massively Distributed IaaS Facilities. Research Report RR-8779, Inria, September 2015.

[MKB18] Redowan Mahmud, Ramamohanarao Kotagiri, and Rajkumar Buyya. *Fog Computing: A Taxonomy, Survey and Future Directions*, pages 103–130. Springer Singapore, Singapore, 2018.

[MTK06] Athina Markopoulou, F. Tobagi, and M. Karam. Loss and delay measurements of Internet backbones. *Computer Communications*, 29(10):1590 – 1604, 2006. Monitoring and Measurements of IP Networks.

[NSS10] Erik Nygren, Ramesh K. Sitaraman, and Jennifer Sun. The akamai network: A platform for high-performance internet applications. *SIGOPS Oper. Syst. Rev.*, 44(3):2–19, August 2010.

[Pas12] Andrea Passarella. A survey on content-centric technologies for the current internet: Cdn and p2p solutions. *Computer Communications*, 35(1):1 – 32, 2012.

[PDÉ+14] Dimitri Pertin, Sylvain David, Pierre Évenou, Benoît Parrein, and Nicolas Normand. Distributed File System based on Erasure Coding for I/O Intensive Applications. In *4th International Conference on Cloud Computing and Service Science*, Barcelone, Spain, April 2014.

[PFTK98] Jitendra Padhye, Victor Firoiu, Don Towsley, and Jim Kurose. Modeling TCP Throughput: A Simple Model and Its Empirical Validation. *SIGCOMM Comput. Commun. Rev.*, 28(4):303–314, October 1998.

[PHS+16] L. Pacheco, R. Halalai, V. Schiavoni, F. Pedone, E. Rivière, and P. Felber. Globalfs: A strongly consistent multi-site file system. In *2016 IEEE 35th Symposium on Reliable Distributed Systems (SRDS)*, pages 147–156, September 2016.

[PSW+15] Z. Pang, L. Sun, Z. Wang, E. Tian, and S. Yang. A survey of cloudlet based mobile computing. In *2015 International Conference on Cloud Computing and Big Data (CCBD)*, pages 268–275, November 2015.

[SMK+01] Ion Stoica, Robert Morris, David Karger, M. Frans Kaashoek, and Hari Balakrishnan. Chord: A scalable peer-to-peer lookup service for internet applications. *SIGCOMM Comput. Commun. Rev.*, 31(4):149–160, August 2001.

[SSMM14] R. D. Souza Couto, S. Secci, M. E. Mitre Campista, and L. H. Maciel Kosmalski Costa. Network design requirements for disaster resilience in IaaS Clouds. *IEEE Communications Magazine*, 52(10):52–58, October 2014.

[SSX+15] M. Satyanarayanan, P. Simoens, Y. Xiao, P. Pillai, Z. Chen, K. Ha, W. Hu, and B. Amos. 0edge analytics in the internet of things. *IEEE Pervasive Computing*, 14(2):24–31, April 2015.

[THS10] Osamu Tatebe, Kohei Hiraga, and Noriyuki Soda. Gfarm grid file system. *New Generation*

Computing, 28(3):257–275, July 2010.

[TT05] Minh Tran and Wallapak Tavanapong. On using a cdn's infrastructure to improve file transfer among peers. In Jordi Dalmau Royo and Go Hasegawa, editors, *Management of Multimedia Networks and Services*, pages 289–301, Berlin, Heidelberg, 2005. Springer Berlin Heidelberg.

[TZV+11] F Tao, L Zhang, V C Venkatesh, Y Luo, and Y Cheng. Cloud manufacturing: a computing and service-oriented manufacturing model. *Proceedings of the Institution of Mechanical Engineers, Part B: Journal of Engineering Manufacture*, 225(10):1969–1976, 2011.

[WBMM06] Sage A. Weil, Scott A. Brandt, Ethan L. Miller, and Carlos Maltzahn. Crush: Controlled, scalable, decentralized placement of replicated data. In *Proceedings of the 2006 ACM/IEEE Conference on Supercomputing*, SC '06, New York, NY, USA, 2006. ACM.

[WLBM07] Sage A. Weil, Andrew W. Leung, Scott A. Brandt, and Carlos Maltzahn. Rados: A scalable, reliable storage service for petabyte-scale storage clusters. In *Proceedings of the 2Nd International Workshop on Petascale Data Storage: Held in Conjunction with Supercomputing '07*, PDSW '07, pages 35–44, New York, NY, USA, 2007. ACM.

[WSJ17] Martin Wollschlaeger, Thilo Sauter, and Juergen Jasperneite. The future of industrial communication: Automation networks in the era of the internet of things and industry 4.0. *IEEE Industrial Electronics Magazine*, 11(1):17–27, 2017.

[WW17] Jake Wires and Andrew Warfield. Mirador: An active control plane for datacenter storage. In *Proceedings of the 15th Usenix Conference on File and Storage Technologies*, FAST'17, pages 213–227, Berkeley, CA, USA, 2017. USENIX Association.

[YZX+10] Zhi Yang, Ben Y. Zhao, Yuanjian Xing, Song Ding, Feng Xiao, and Yafei Dai. Amazingstore: Available, low-cost online storage service using cloudlets. In *Proceedings of the 9th International Conference on Peer-to-peer Systems*, IPTPS'10, pages 2–2, Berkeley, CA, USA, 2010. USENIX Association.

[ZMK+15] Ben Zhang, Nitesh Mor, John Kolb, Douglas S. Chan, Nikhil Goyal, Ken Lutz, Eric Allman, John Wawrzynek, Edward Lee, and John Kubiatowicz. The cloud is not enough: Saving iot from the cloud. In *Proceedings of the 7th USENIX Conference on Hot Topics in Cloud Computing*, HotCloud'15, pages 21–21, Berkeley, CA, USA, 2015. USENIX Association.

List of terms

Biographies

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 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.

November 2019

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.