Performance Analysis of Object Store Systems in a Fog and Edge Computing Infrastructure

Bastien Confais, Adrien Lèbre, Benoît Parrein

Performance Analysis of Object Store Systems in a Fog and Edge Computing Infrastructure

Bastien Confais[1], Adrien Lebre[2], and Benoît Parrein[3]

[1] CNRS, LS2N, UMR 6004,
Polytech Nantes, France
bastien.confais@univ-nantes.fr
[2] Inria, LS2N, UMR 6004,
Institut Mines Télécom Atlantique, Nantes, France
adrien.lebre@inria.fr
[3] Université de Nantes, LS2N, UMR 6004,
Polytech Nantes, France
benoit.parrein@polytech.univ-nantes.fr

Abstract. Fog and Edge computing infrastructures have been proposed as an alternative to the current Cloud Computing facilities to address the latency issue for some applications. The main idea is to deploy smaller data-centers at the edge of the backbone in order to bring Cloud Computing resources closer to the end-usages. While a couple of works illustrated the advantages of such infrastructures in particular for Internet of Things (IoT) applications, the way of designing elementary services that can take advantage of such massively distributed infrastructures has not been yet discussed. In this paper, we propose to deal with such a question from the storage point of view. First, we propose a list of properties a storage system should meet in this context. Second, we evaluate through performance analysis three "off-the-shelf" object store solutions, namely Rados, Cassandra and InterPlanetary File System (IPFS). In particular, we focus (i) on access times to push and get objects under different scenarios and (ii) on the amount of network traffic that is exchanged between the different geographical sites during such operations. We also evaluate how the network latencies influence the access times. Experiments are conducted using the Yahoo Cloud System Benchmark (YCSB) on top of the Grid'5000 testbed. Finally, we show that adding a Scale-Out NAS system on each site improves the access times of IPFS and reduces the amount of traffic between the sites when objects are read locally by reducing the costly DHT access. The simultaneous observation of different Fog sites also constitutes the originality of this work.

1 Introduction

The advent of smartphones, tablets as well as Internet of Things (IoT) devices revolutionized the ways people are consuming IT services. Lots of applications take advantage of the Internet and Cloud Computing solutions to extend devices' capabilities in terms of computations as well as storage. However, reaching data centers (DCs) operated by giant actors such as Amazon, Google and

Microsoft implies significant penalties in terms of network latency, preventing a large amount of services to be deployed [59]. The Fog Computing paradigm [9] has been proposed to overcome such a limitation: dedicated servers are deployed in micro/nano DCs geographically spread at the edge of the network so that it becomes possible to execute latency dependent applications as close as possible to the end-usages and keep non sensitive ones in traditional Cloud DCs. Previously, Content Delivery Networks (CDN) used a similar approach to reduce access time to data for the end users [39].

Also known as Edge Computing, the advantages of such infrastructures have been described through several scenarios [2, 20]. In this paper, we propose to check if storage systems designed for Cloud infrastructures may be used in a Fog environment. Concretely, we discuss an empirical analysis of three storage systems with the ultimate goal of delivering a system such as the Simple Storage Service (S3) of Amazon. S3 is one of the most used services offered by Amazon and a building block for hundreds of Cloud services [38]. We believe that providing such a storage service for Fog/Edge Computing infrastructures can pave the way toward new services as well as IoT applications. In terms of use cases, we have in mind to provide to a mobile end user a perfect seamless storage experience towards different residential sites *e.g.*, home, public transportation, office. The three storage systems, we studied are Rados [52] which is the object storage module of the Ceph project, Cassandra [30], a high performance key value store and InterPlanetary File System (IPFS) [7], an object store which uses the concepts brought by BitTorrent protocol. We selected these three systems because *(i)* they do not rely on a central server and *(ii)* they propose software abstractions for the definition of geographical *sites*. They also propose strategies to enable users to place data near the users, mitigating the traffic exchanged between each site and reducing the impact each site may have on the others.

The contributions of our work are *(i)* the definition of Fog/Edge computing model with a list of dedicated requirements for storage service, *(ii)* an overview of the three evaluated storage systems namely Rados, Cassandra and IPFS with a specific Fog adaptation, *(iii)* a deep performance analysis of the three systems in a Fog context. We evaluate the performance in case of local and remote access. More precisely, we measure access times and amount of network traffic exchanged between the sites. We determined for each system what are the key settings that have an influence on the access times. A last contribution is *(iv)* to show that the impact of using a DHT in IPFS can be partially prevented by using a scale out NAS system deployed on each site. We prove that such a system is able to mitigate the amount of inter-sites network traffic as well as the access times when clients read objects stored locally. Experiments have been conducted on top of Grid'5000 [6] by considering several data manipulation scenarios leveraging Yahoo Cloud Service Benchmark (YCSB) [16], a well-known benchmark tool particularly designed to benchmark object stores [3, 4]. Moreover, we used a benchmark we developed to measure the access times of each object. Similar to YCSB, this benchmark enabled us to see how objects are accessed in IPFS.

The remaining of the paper is organized as follows. Section 2 defines the Fog/Edge computing model we consider and gives a list of characteristics a data store service should have in such a context. Section 3 presents an overview of the three storage systems. Evaluations are discussed in Section 4. In Section 5, we introduce a coupling between IPFS and a Scale-Out NAS. Section 6 discusses the related works. Finally, Section 7 concludes this study and highlights some perspectives.

2 Fog and Edge Computing Model

In this section, we present the Fog/Edge architecture we are considering. Then, after listing some use cases the Fog can benefit, we present a list of characteristics we claim an object storage system should have in such a context.

2.1 Fog/Edge networking

Industrials[1] as well as academics [9, 22, 23] argue in favor of a new model of distributed computing composed of IT resources spread from the Cloud to the Extreme Edge. Such an infrastructure 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 tradeoff between distance and power capabilities of IT resources [8, 23]. Moreover, Fog sites can complement each other to satisfy the needs between user's devices and Cloud Computing centers [12]. According to Bonomi *et al.* [9], the Fog infrastructure is organized in several layers with a Cloud infrastructure at the top. We consider all the sites of Fog be part of the same layer.

Figure 1 illustrates such a description. The Fog platform is composed of a significant number of sites that can be geographically spread over a large area. We also argue that users may benefit that some sites of Fog may be mobile. Placing a Fog facility in a train for example, allows the users to stay connected to the same site of Fog with a stable latency despites of the mobility of the train. Each site hosts a limited number of servers that offer storage and computing capabilities. Nonetheless, according to Firdhous *et al.* [22], Fog nodes should have enough resources to handle intensive user requests. End-users devices (smartphones, tablets, laptops) as well as IoT devices can reach a Fog site with a rather low latency. Devices located in the Edge are directly connected to the Fog whereas those located in the Extreme Edge have to cross a local network to reach the Fog. Fog sites are interconnected and the single distributed storage system makes them transparent from the users point of view.

[1] `https://www.openfogconsortium.org/`

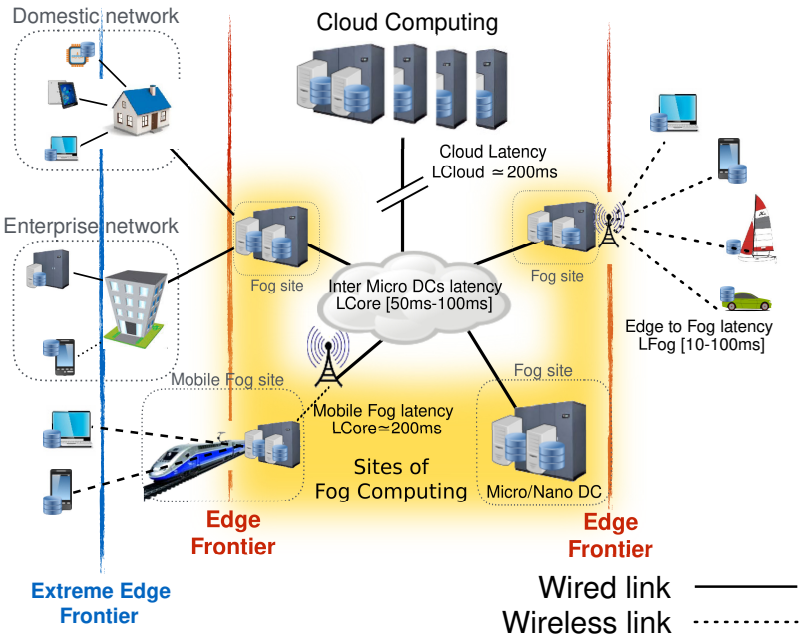


Fig. 1: Overview of a Cloud, Fog and Edge infrastructure.

We consider that the latency between Fog sites (noted L_{Core}) is up to 50 ms (mean latency of a Wide Area Network link [34]) for a static site of Fog and is up to 200 ms for a mobile one. We also consider the latency between users and their site (noted L_{Fog}) is comprised between 10 ms and 100 ms (latency of a local wireless link [29, 46]). The latency to reach a Cloud infrastructure (noted L_{Cloud}) from the clients is important (about 200 ms) [22,45] and moreover unpredictable [59].

2.2 Fog and Edge Computing: use cases

Many use cases have been proposed for the Fog. Yannuzzi *et al.* [54] show the Cloud Computing cannot address all the use cases and discuss the tradeoff the Fog has to solve, especially between the mobility support, the access times constraints and the amount of data that needs to be stored/computed. They show the Cloud cannot satisfy this tradeoff and a dual approach Fog/Cloud is necessary. A lot of papers propose the smart vehicles or the smart traffic light [8,9,55] as use cases that can benefit the Fog architecture. Hong *et al.* [25] have developed a programming model for the Fog and they used it to monitor vehicle traffic. Tang *et al.* [47] propose to use the Fog in the context of smart city, especially to monitor pipelines. We also found some use cases in the health field. Dubey *et al.* [20] use the Fog to make quick decisions from the values collected from sensors. Zao *et al.* [58] compute EEG pattern detection on the Fog. Finally, Fog can be used for networking services. Vaquero *et al.* [49] deploy Network Function

Virtualisation (NFV) in the Fog. Virtual machines have the role of network appliances such as router, firewall and so on. The paper argue, the Fog can reduce the load of some network paths, especially the ones reaching the Cloud. Yi *et al.* [56] propose among a list of use cases, to use the Fog for data caching. In this paper, we study how a datastore system can be developed to benefit from Fog/Edge specifics.

2.3 Storage Requirements

Our objective is to study how a storage service such as a S3 object store system should be designed to deal with a Fog Computing infrastructure. The first requirement for such a system is the scalability that is required to consider the huge number of sites the Fog is composed of. Contrary to the common distributed file systems, object stores are not concerned by the problematic of storing the global namespace in a scalable way and therefore are more adapted to a Fog Computing infrastructure. We advocate that a Fog Computing storage service should meet the following properties:

- data locality (enabling low access time);
- network containment between sites;
- possibility to access data in case of service/network partitioning;
- support for users mobility;
- scalability with a large number of sites, users and objects stored.

Low access time is the main characteristic behind the motivation for the Fog paradigm. The idea of **data locality** is to favor local accesses each time it is possible. Each `put` into the storage service should be handled by the closest site, assuming that the closest site can deliver the best performance. The data locality property also adresses security and privacy concerns [57].

Network containment is the idea that an action on one site does not impact the other sites negatively. In other words, if one site faces a peak of activity, the performance of other sites should not change. We believe that such a criterion can be delivered by mitigating data transfers between sites each time an operation is performed. The verification of this property assumes the simultaneous observations among all the Fog sites.

The third feature is related to the **partitioning of the storage service** that can occur each time one site is disconnected from the other ones. While replication strategies between sites can ensure data availability, we claim that a Fog storage service should be able to run, at least, in a degraded/disconnected mode in order to tolerate local accesses and provide appropriate mechanisms to reconsolidate the service once the disconnection is completed.

Mobility support is another property we have identified. The idea is to enable data to follow transparently its usages. To illustrate such a feature, you can imagine a user moving from one radio base station to another one. In such a situation, the storage service should be able to relocate solicited data in a transparent manner from the previous site to the new one. Such a transparent

relocation of data will mitigate remote accesses and allows the system to satisfy the aforementioned low access characteristic.

Scalability is the last property. The system has to scale to a large number of sites, with a lot of clients connected to these sites, storing a lot of objects. Performance should not be impacted by the increase of the number of sites.

According to the Brewer's theorem [11], a storage system cannot have strong consistency if each request is answered in a finite time and if the system supports the service partitioning. For this reason, we underline that discussing consistency model of the storage service we target is behind-the-scope of this first study. For the moment and for the sake of simplicity, we consider objects like files and documents of one user with no parallel and concurrent accesses.

3 Off-the-shelf Distributed Storage Systems

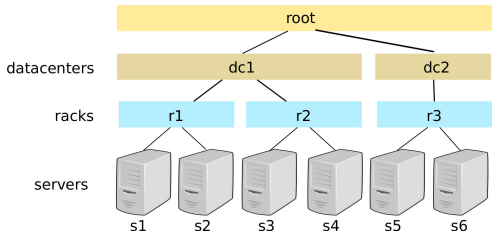
Due to the complexity of modern distributed storage systems, it is important to evaluate if any of the existing systems may be used. Distributed Hash Table (DHT), gossiping and hashing are the main mechanisms used by distributed object stores in charge of storing the location of the objects. Among the different solutions that are available, we selected Rados [52], Cassandra [30] and IPFS [7]. We chose these systems because they provide software abstractions that enable the definition of areas that can be mapped to geographical sites, and thus may be adapted to a Fog Context.

Distributed file systems such as PVFS [13], Lustre [19], HDFS [44], RozoFS [40] and "other" WANWide-like proposals [26, 48] have not been selected because they are not appropriated to the Fog Computing infrastructure we target. They are all designed around the concept of an entity in charge of maintaining the storage namespace in a "centralized" manner, preventing conceptually to cope with Fog Computing requirements we previously described. Especially, the network containment property cannot be satisfied because writing or reading on a local site requires remote access to the metadata server. This justify our coupling of IPFS with a Scale-Out NAS system in Section 5. We present in the following paragraphs, an overview of the three selected storage systems and analyze quantitatively whether and how they can fit to the Fog/Edge context.

3.1 Rados

Rados [52] is an object distributed storage solution which uses the CRUSH algorithm [51] to locate data in the infrastructure without requiring any remote communication from the clients. Conceptually speaking, this constitutes the main interest of Rados for Fog storage solutions.

General Overview Rados uses two kinds of nodes: Object Storage Daemons (OSD) and Monitors. The formers are used to store data whereas the latters maintain a tree (*i.e.*, the "clustermap") describing the cluster's topology. Among the monitors, one is elected as the master and is in charge of maintaining a



(a) – Example of a clustermap

1. select the "root" node
2. select 2 "datacenters"
3. select 1 "rack"
4. select 1 "server"

(b) – Example of a placement rule

Fig. 2: Example of a clustermap (a) and a placement rule (b) placing two replicates of the data in two servers located in two different datacenters.

consistency view of the clustermap (**the Paxos algorithm** [31] is used to guarantee that there is only one master monitor). Before accessing data, each client retrieves the **clustermap** from one monitor. The clustermap is used by the CRUSH algorithm to locate objects.

In addition to the clustermap, the CRUSH algorithm relies on placement rules that describe how selecting the "n" leaf nodes from the clustermap used to store the object. Figure 2(a) depicts an example of a clustermap. It describes a topology containing two data centers with two racks in the first DC and one rack in the second DC. Each rack contains two servers. Figure 2(b) shows a placement rule to force two replicas to be located in two different data centers.

The location of an object is performed by hashing its name (key) in order to determine the placement group it belongs to. The placement groups (PG) are sets of objects for which all replicas are placed on the same devices. CRUSH is then executed on the identifier of the "placement group" to determine the location of the object. With the clustermap given in Figure 2(a) and the rule in Figure 2(b), the placement is performed as follows: *(i)* the root node of the clustermap is selected; *(ii)* 2 children nodes are chosen: in our example "dc1" and "dc2" are automatically selected because they are the only candidates; *(iii)* one rack is selected is each DC, for example "r1" and "r3" in our example; (iv) finally, one server is selected in each rack, *e.g.*, the two replicas will be placed on the servers "s5" and "s2".

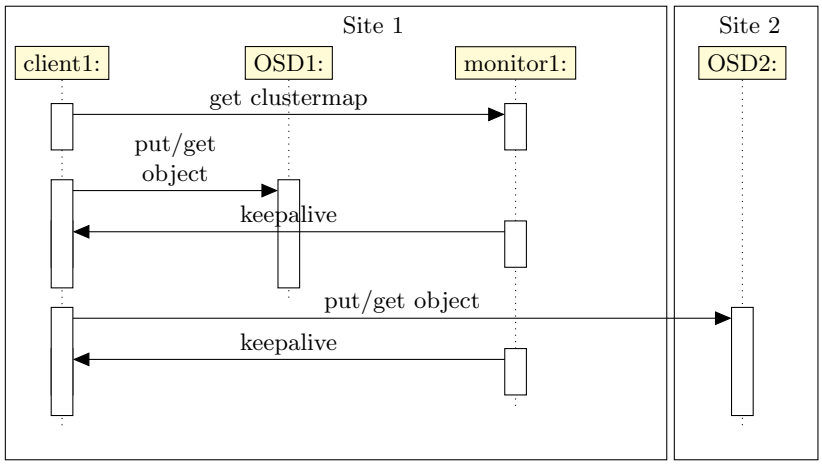
Figure 3 shows the sequence diagram of the message exchanges we observe in a Rados deployment in a multi-sites context, respectively from a client (a), an OSD (b) and a monitor (c) point of view. In addition to the exchanges between the monitors, Rados uses a large number of keepalives messages between the different nodes. This enables the system to swiftly react in case of failure. We will reuse this Figure in the next section explaining how Rados can be adapted in a Fog infrastructure.

Finally, we highlight that when the network is partitioned, only the data located in the partition where a master monitor can still be elected, is available. This partition does not necessarily exist if none of the parts contains a majority

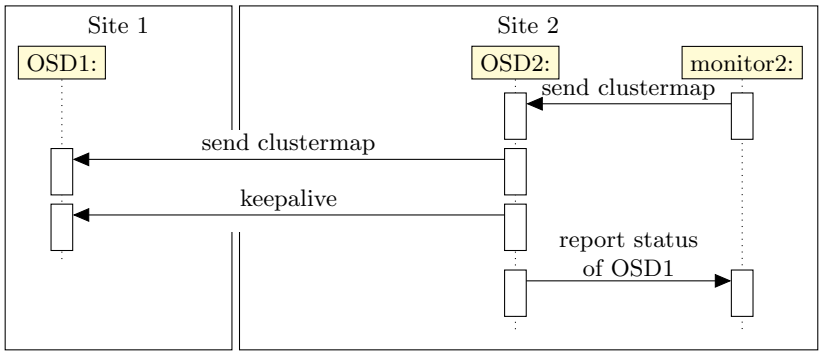
of monitors. In other words, clients belonging to the partition containing a master monitor can access the clustermap and thus any object that is reachable. On the other partitions, because clients cannot get the clustermap they cannot locate and cannot access any object.

Fog Considerations Rados allows administrators to organize objects within pools. Each pool is associated to settings like a replication factor and a "placement rule" describing where the replicas of the objects belonging to the pool are stored. Each "pool" defines a **namespace**, thus, to write or read an object, clients must provide the name of the pool. To retrieve an object, users must know the couple (pool, object_name).

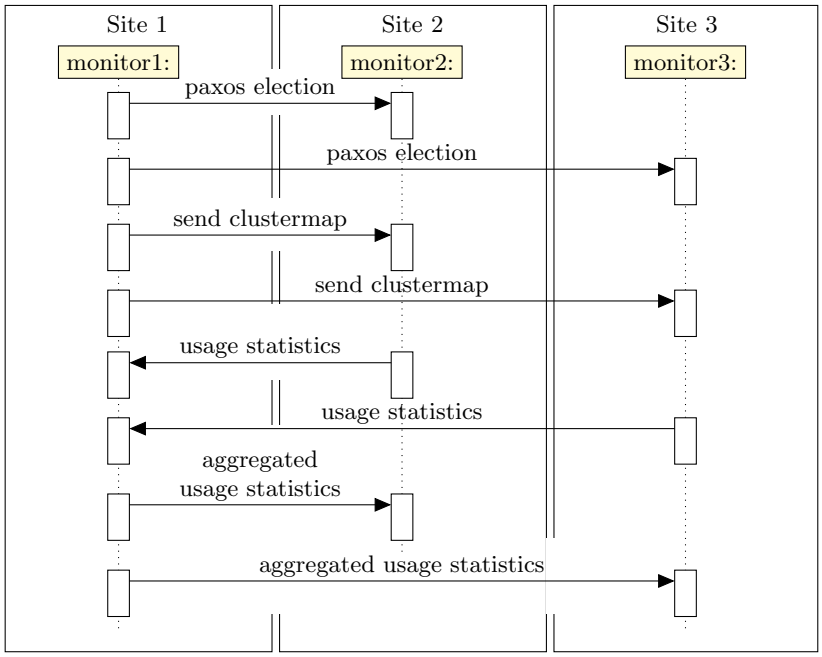
To favor the data locality property we introduced in Section 2.3, we propose to use placement rules to constraint objects of a particular pool to be located in one specific DC (*i.e.*, one specific site). Figure 4 presents some placement



(a) – From **a client** point of view



(b) – From **an OSD** point of view



(c) – From **a monitor** point of view – the monitor on the site 1 is elected, its clustermap is used by all nodes

Fig. 3: Sequence diagrams of the network traffics observed in Rados.

rules associated to pools. With a pool associated to each site, the drawback is the objects of a user cannot be moved easily. If a user moves, its objects have to be placed in another pool located in another site. So, the user must know in which pool it stores its data. Moreover, this approach implies that all data movements are initiated by the user. For example, an administrator cannot make the decision to move the objects from one location to another one because the user will not be able to find them. The user will continue to use the previous pool which contained its objects and not the new one. A solution for this problem could be to create a pool per user. By changing the placement rule the pool is using, data are automatically relocated to fit the new placement rule. However, this approach is not scalable when the number of users becomes significant as it will considerably increase the size of the clustermap (the list of pools and the placement rules being stored in the clustermap, its size will grow according to the number of pools/users leading to important network overheads each time the clustermap will be exchanged). The limitation in terms of mobility is also another drawback of this approach: because a pool can only be attached to one site, a user cannot store its data across distinct locations. Each time a user moves from one location to another one, it has to request the relocation of its whole

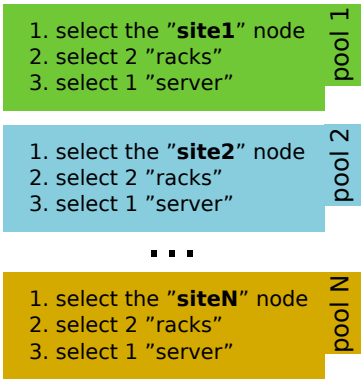


Fig. 4: Example of placement rules associated to pools adapted to a Fog context. The site storing the replicas is specified in each rule. In this example, all objects belonging to a "pool N" are stored on the "site N".

pool. In other words, there is no mechanism that enables users/administrators to relocate only the sollicited objects. The larger the pool, the more expensive the relocation operation is, facing potentially ping/pong effects where data goes back and forth between sites.

From the Figure 3 the inter-sites overhead is a mix of Paxos' messages, distributions of clustermap, and usage statistics sent between the monitors. The inter-sites covers also some report of OSD status. To mitigate as much as possible this overhead, we propose to place one monitor per site. This minimizes the report of OSD status between the sites as well as the overhead related to the clustermap retrievals but it maximizes the amount of usage statistics sent between monitors. We do not have checked if the amount of network traffic between the sites becomes less important when fewer monitors are used. But for sure, with fewer monitors, the amount of network traffic vary with the number of clients because some of them have to contact a remote monitor to get the clustermap. We point out the placement of the monitors does not affect the keepalive sent from the OSDs to their neighbours, as depicted in Figure 3(b). Having one monitor per site avoids the status reporting from the OSD to the monitor becomes an inter-sites traffic. Nevertheless, **the Paxos protocol limits the number of monitors that can be used in Rados and therefore the scalability of the system**.

To conclude, the main adaptations we propose for Rados to fit to the Fog requirements are:

1. Creating a "pool" per user with placement rule for data locality and for mobility support.
2. Placing a monitor per site to limit exchanges of metadata;

We now consider Cassandra as a second possible solution.

3.2 Cassandra

Cassandra [30] is a key value store system that uses gossip and hashing to place the data.

General Overview Cassandra is organized as a **one-hop Distributed Hash Table (DHT)**. The set of values the object keys are defined on is divided into ranges that are distributed among the nodes composing the system. A **gossip protocol** is used to distribute the topology of the system, the status of the nodes and the ranges affected to them. Each second, each host sends a gossip packet to another one randomly selected. Once gossiped data is received, storage nodes can locate any object without any extra communication. They simply hash the object name and look for using the gossiped data, the node which is responsible for the key.

A quorum, specified by users defines the number of replicas that has to be read or written for each request to validate the operation. Depending on the values used in this quorum, Cassandra can provide different levels of consistency. This quorum provides a trade-off between access time and data consistency. When the network is partitioned, data can be accessed as long as the client is able to retrieve as many replicas as required by the quorum. As an example, if they are two replicas, one in each network partition and the quorum specified is to retrieve the two replicas for strong consistency, the request of the client will fail because the quorum cannot be satisfied.

Moreover, Cassandra exposes the same notion of "pools" as the one proposed by Rados. Entitled "keyspaces", they define **different namespaces**. Each "keyspace" is associated to a replication factor and to a "replication strategy" defining where the replicas are stored. Like Rados, users have to specify the keyspace's name they want to use when they perform an operation.

The major exchanges related to the Cassandra protocol are illustrated on Figure 5: Clients retrieve the topology from the server they connect to. Then they open a connection with some nodes composing the cluster. Several strategies are proposed to select the server used to send a request. By default, requests are balanced in a round-robin way. Clients send requests alternatively to the servers they are connected to. Each request is then forwarded to the server, which has to handle it (the server is determined based on the aforementioned strategy).

Fog Considerations As described Cassandra proposes different strategies to locate data throughout the infrastructure. The "NetworkTopologyStrategy" is a placement strategy that specifies how many replicas should be stored on each data center (in our case on each site). For the purpose of our analysis, we configure this strategy in order to have only one copy of an object in a particular site throughout the whole system. Such a strategy enables Cassandra to mitigate the traffic between sites (the different versions of a given object are all located in the same site, limiting the traffic related to synchronize them) and provides the data locality criteria we want to favor. We highlight that having one copy of the data also guarantees strong consistency like proposed by Rados.

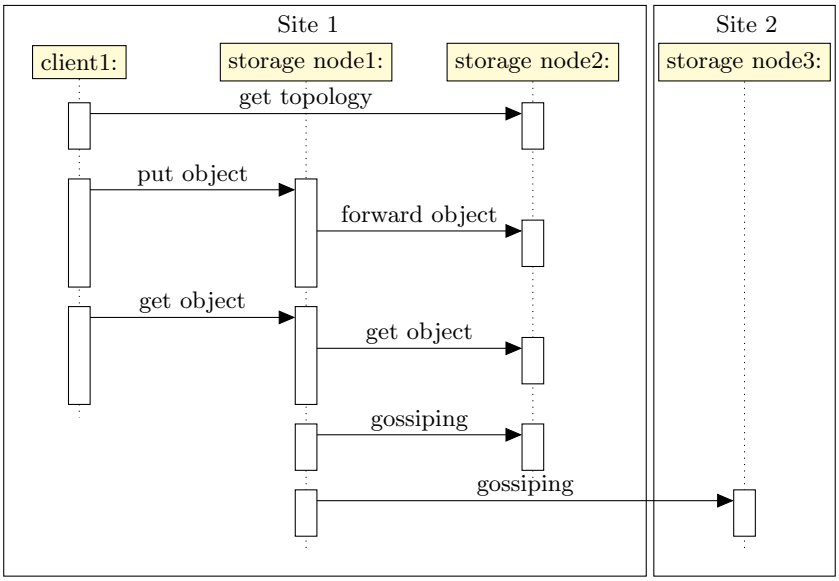


Fig. 5: Sequence diagrams of the network traffics observed in Cassandra.

With "keyspaces", we have exactly the same problematic with Cassandra we have with the "pools" of Rados. We have to determine how to create the "keyspaces" in the cluster. If a "keyspace" per site is created, the user will have to remember in which "keyspace" each object is stored. Moreover, in case of mobility, the "keyspace" storing the objects will change. With a "keyspace" per user, it avoids users to remember where objects are located but it may not be scalable because "keyspace" list, associated to their replication strategy is replicated on all the nodes. Moreover, in Rados, "pools" settings are propagated from the monitors to the OSDs. In Cassandra, monitors do not exist. The list of "keyspaces" and the replication strategies associated to them is spread on all the storages nodes. To keep this list conistent between the nodes, the operation of creating a "keyspace" requires a lock on all the nodes. The creation of "keyspaces" does not work in a failure context, if some nodes are unavailable.

Moreover, the "replication strategy" associated to a "keyspace" can be modified. However, similar to Rados, the relocation requires explicit administration operations to redefine the replication strategy. Thus, **the mobility is not well-supported by Cassandra**.

Regarding the network traffic in Figure 5, the gossip messages are the only overhead that goes throughout the different sites. This traffic is independent on the sites activities. But the network traffic inside the sites can vary depending on the size of the sites. Because of the forwarding mechanism previously described, the more nodes a site has, the more the traffic inside the site. With a lot of nodes clients will send their requests to the node which stores the needed object with

a lower probability. Thus, the forward mechanism will be more used increasing the amount of network traffic inside each site.

To conclude, the main adaptations we propose for Cassandra are:

1. Creating a "keyspace" per user;
2. Using the "NetworkTopologyStrategy" placement strategy for data locality.

3.3 InterPlanetary File System

InterPlanetary File System [7] has been built on the **BitTorrent protocol** [32] and a **Kademlia DHT** [35]. BitTorrent and Kademlia are both being well-known protocols for their ability to scale to a large number of nodes. While the BitTorrent protocol is used to manipulate objects between the different peers of the system in an efficient manner, the Kademlia DHT is in charge of storing the objects' location. We underline that it is only the locations and not the content of the objects that are stored in the DHT. Such a management of the data locations is an important difference in comparison to Rados and Cassandra that have designed dedicated mechanisms to locate objects without extra communication from the clients point of view.

IPFS uses **immutable objects**. Modifying an existing object lead to creating a new one. Because objects are immutable, it is easier to maintain the consistency between all replicas. Moreover, the BitTorrent protocol that is used to pull the data enables IPFS to retrieve the same object from several sources simultaneously [41].

Contrary to Rados and Cassandra, users cannot choose the object's names. The name of an object depends on its content. Indeed, the name of an object is a checksum of the object. That is also a consequence of the object immutability.

Figure 6 shows the major message exchanges of IPFS. When a client wants to put an object, the client sends the object to a node. This node saves the object locally and then puts the location of the object in the Kademlia DHT. Reciprocally, when a client wants to get an object, it has to contact one peer of IPFS. This peer checks if it stores the object locally. In this case, the object is directly send to the client. Otherwise, the IPFS node will use the Kademlia DHT to determine the node in charge of delivering the object. Based on the Kademlia reply, the request is forwarded to the correct node. This node will send the object to the initial IPFS peer that will make a copy before serving the client. Thanks to such an approach, **IPFS supports the mobility of data in a native fashion**. This node reports the existence of its new replica in the DHT, so that a future read can be satisfied by this node, closer to the client. Storage nodes send regularly keepalive messages to maintain the DHT. The DHT contains the location of all the replicas of all the objects. In case of remote reading, the DHT is accessed two times: a first time to retrieve the location of the needed object and a second time to update the location, in order to reflect the existence of the new replica.

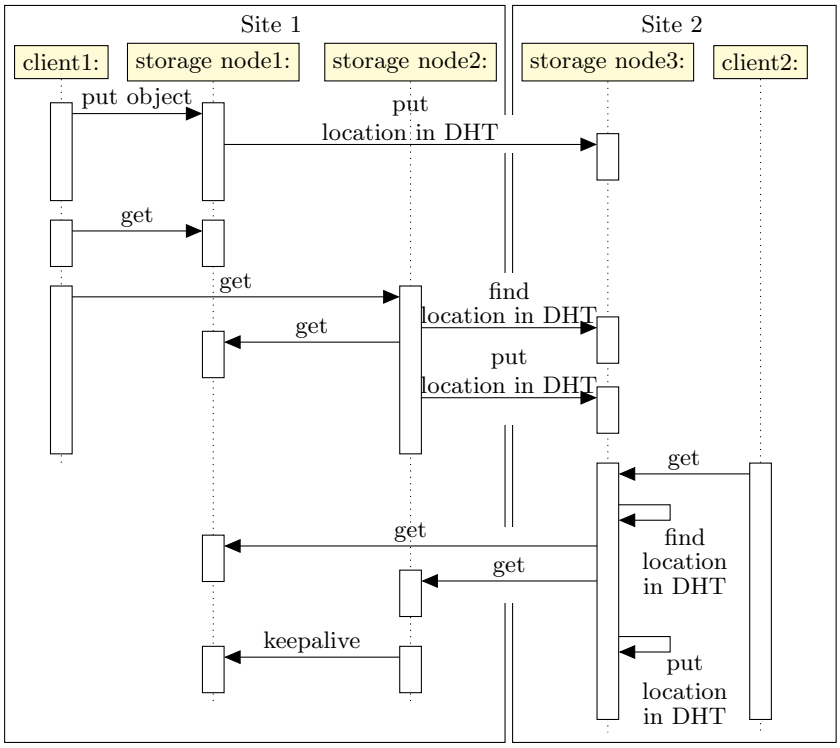


Fig. 6: Sequence diagram of the network traffic observed in IPFS from **a client** point of view. The read from "client2" on "storage node2" can only be performed after the object was relocated on this storage node, at the end of the read from "client1".

Fog Considerations By its design, IPFS favors to store objects locally: only the use of the Kademlia DHT leads to inter-sites traffic. In conclusion, there is no specific adaptation to achieve with IPFS in order to better fit the Fog requirements than the default configuration. IPFS can work partially in disconnected mode as long as both the object location can be found in the Kademlia DHT and the node storing the object is reachable.

3.4 Fog characteristics met for the object stores

Table 1 summarizes how do Rados, Cassandra and IPFS fit the requirements that have been defined in Section 2.3. As discussed previously, the data locality property in Rados and Cassandra can lead to scalability problems if a pool or a keyspace is created for each user. Also, these two systems support the mobility partially because they require an administration intervention. Rados supports

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

Table 1: Summary of Fog characteristics *a priori* met for 3 different object stores.

partially the network containment characteristic because its use of the CRUSH algorithm. Finally, the Paxos algorithm does not make Rados scalable.

In the next part, we perform performance evaluation of the three systems but Rados can only be used as a reference point because of it lacks of scalability and functionality in a Fog context.

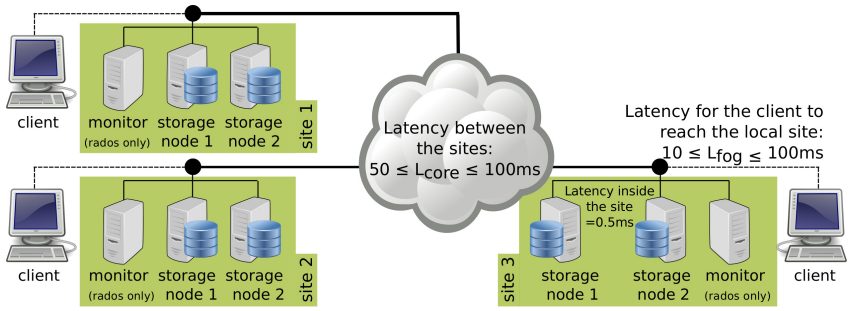
4 Benchmark of Rados, Cassandra and IPFS

This section discusses the different evaluations we performed. Experiments settings are given in Section 4.1. Section 4.2 analyzes a first set of experiments that aimed to evaluate the network containment property. A second set of experiments that enabled us to investigate remote access performance (*i.e.*, when objects are accessed from a remote Fog site) is discussed in Section 4.3.

4.1 Material and method

The material and method we used for the benchmark are described in the following.

Testbed description Experiments have been performed on the Grid'5000 testbed [6], using the "Paravance" cluster hosted in the city of Rennes (Dell powerEdge, Intel Xeon, 16 cores, 128 GB RAM, 10 Gbps Ethernet).



(a)

Fig. 7: Topology used to deploy Rados, Cassandra and IPFS in a Fog environment. Monitors are only deployed for Rados.

The Fog architecture we emulated as well as the way we deployed the different systems are presented in Figure 7. Each Fog site is composed of two storage servers and one additional server is deployed to deliver the Monitor service for Rados experiments. Finally, each site has its own local client. Latencies between servers is set using the Linux traffic control utility (`tc`). Unless otherwise specified, we use $L_{Fog} = 10\,\text{ms}$ and $L_{Core} = 50\,\text{ms}$. The latency between the servers located on a same site is considered low and has been set to $0.5\,ms$. The throughput of the network links is set to 10 Gbps, both for intra and inter-sites links. More details about the physical topology are given on the Grid'5000 website: `https://www.grid5000.fr/mediawiki/index.php/Rennes:Network`.

There are two reasons to simplify the network model by considering only the latency in this work. The first reason is the network latency is a good measure of the geographical distance. According to Dabek *et al.* [18] by example, "inter-host RTT is dominated by geographic distance". We consider the geographical distance between the sites is more important than the distance for a client to reach its site. Therefore, we set $L_{Core} > L_{Fog}$. The second reason is network impairment can be considered as an increase of the latency. Indeed, according to the Nielsen's law [36] and more recent studies [24] the latency is a predominant criterion that results from network impairments such as network congestion, packet losses, retransmission and so on. Finally, according to Padhye *et al.*, increasing the latency has also an impact on the throughput achievable by TCP [37].

The definition of placement strategies such as described in Sections 3.1 and 3.2 has been achieved respectively for Rados and Cassandra. This enabled us to write all data locally (we remind that data is written on the server the client contacts for IPFS). We configured the replication strategy of each system in order to get only one copy of each object. This modification enabled us to get a minimal boundary of access times. We believe the replication for fault tolerance can only increase the access times. We assume that a given technic of replication may impact the access times of the three systems in the same way. Therefore, to get the best access times as possible, we disabled it. We also modified the code of IPFS to avoid the intensive replication strategy it uses for metadata: by default for each object, it inserts ten copies of the corresponding metadata in the Kademlia DHT. We disabled this mechanism in order to get only one copy. Disabling all replications mechanisms allowed us to remove any possible bias in our analysis of pros/cons of the storage protocols used in each of the three systems.

Finally, the metrics we measured are the time taken to perform each operation on each site and the amount of network traffic sent between them on the experiment period. All file system partitions are unmounted between each experiment to avoid cache effects between consecutive executions.

YCSB benchmark and workloads The Yahoo Cloud System Benchmark [16] (YCSB) has been used to evaluate the three systems. YCSB proposes different workloads depending *(i)* on the amount of data written, *(ii)* the size of the objects, *(iii)* the proportions of `read`, `update` and `delete` operations and also

(iv) on how the objects selection is achieved. Our experiments performed write and read accesses of different sizes: clients connect the storage systems, execute write and then read operations to finally close their connection. Neither update nor delete operations are performed. The object sizes have been chosen to be representative to real scenarios [5]. Because different object stores can be suited for different object sizes, we chose to perform our tests using three sizes: 256 KB, 1 MB and 10 MB. Objects sizes of 256 KB correspond to online gaming and web hosting whereas 10 MB can be the size of object used in enterprise backup. 1 MB is an intermediate value. The number of objects used is varying from 1 to 100 per site to show how the systems react under a high load. The number of threads used by YCSB is equal to the number of objects so that all objects are written in a parallel way. In the reading phase, each object is read once and only once thanks to the "sequential" request distribution provided by YCSB. We did not use the default "zipfian" or "uniform" distributions because an object can be read several times, favoring IPFS due to the automatic relocation of data. We point out that the "sequential" distribution does not mean the objects are read sequentially by the client. It means that function returning to the different threads the name of the objects they have to read, this function reads the list of the objects sequentially.

We considered scenarios using 1, 7 and 11 sites simultaneously (*i.e.*, an instance of YCSB is launched on each client of each site). Although Fog environment can be composed of a more significant number of sites, we highlight that performing experiments up to 11 sites is enough to identify several issues the storage systems face to in this context. Each experiment has been performed at least 10 times (10 trials) to get stability in results. Unless precised, the standard deviation is not presented in our discussion as it corresponds in most cases to few hundredths of seconds. Gathered access times correspond to the time to write or read one object (*i.e.*, the time we discuss does not take into account the establishment and closing of connections). This choice is mainly due to the Cassandra client that requires a significant amount of time to open and close connections with the Fog site.

Last but not the least, we highlight that we had to implement a module for IPFS[2]. This module uses the "java-ipfs-api" library proposed by the IPFS developers. To prevent any bias in our study, the IPFS module sends requests in a random way between the servers of the site. This means that one object can have been written on one server and then a read request can be sent to the other server of the site. This behavior is required to avoid accessing the same server and thus never contacting the Kademlia DHT (as described in Section 3.3, if the client contacts the server that stores the requested object, the request can be satisfied without contacting the DHT).

[2] The source code is available at `https://github.com/bconfais/YCSB`

4.2 Local access analysis

All clients (one per site) execute the scenario simultaneously: they write objects on their site and read them. The goal is to evaluate data locality as well as network containment properties of the three systems. This last property has been evaluated by measuring the amount of network traffic exchanged between the site and by analyzing how the object access time is impacted when the number of sites grows.

Writing and reading times Tables 2(a), 2(b) and 2(c) show respectively for Rados, Cassandra and IPFS, the mean times to complete either a write or a read operation. For each of the three systems, we can see that the access times are in the same order of magnitude with respect to the number of sites composing the Fog/Edge infrastructure. As an example, it takes 0.96 seconds per object when 100 objects of 1 MB are written on 1 site and 1.05 seconds using 11 sites for Rados (represented both in bold in the table). Values for Cassandra follow the same trend. By using a mechanism that enables the location of each object without requiring a remote request, the object access time for Rados and Cassandra is not impacted by the number of sites composing the Fog/Edge infrastructure. As discussed later, most of the inter-sites network traffics are sent asynchronously and thus does not impact negatively the performance.

For IPFS, the results are rather surprising as we expected to observe performance degradations for Fog and Edge infrastructures composed of a significant number of sites: the probability to contact a remote site for determining the location of an object increases with respect to the number of sites and accessing the DHT adds a penalty to the object access time. However, we did not observe such an overhead (12.20 vs 11.86 seconds to write one object in the 100×10 MB workload using 3 and 11 sites). Diving into details, we discovered that an object insertion leads to two operations. The first one that consists of storing the object locally on the server is done in a synchronous manner. The second one that consists of pushing the metadata in the Kademlia DHT is performed in an asynchronous manner that is after answering the client. This means that the impact of storing the meta information on a remote server is not visible for write operations. Only read operations can lead to request the DHT is a synchronous manner. However, because of the topology we used, the number of manipulated objects and the number of sites is not important enough to observe performance penalties (3.95 vs 3.93 seconds to read one object in the 100×10 MB workload using 3 and 11 sites). For each client, half of the objects previously created will be retrieved without accessing the DHT. A specific experiment on the overhead of the DHT is discussed in more details in Section 5.

Tables show also that access times are faster for reading than writing for the three systems. There are several reasons for this. The first reason is the hard drives used to store the objects does not have the same performance in writing and in reading. The second reason is that inserting an object requires to compute its hash to determine the node of the DHT which has to store its location. Hash computation also explains that the access time grows with the

	Mean **writing** time (seconds)				Mean **reading** time (seconds)			
	Number \ Size	256 KB	1 MB	10 MB	Number \ Size	256 KB	1 MB	10 MB
1 site	1	0.42	0.78	3.40	1	0.39	0.74	2.53
	10	0.35	0.71	3.24	10	0.34	0.64	2.27
	100	0.35	**0.96**	9.45	100	0.32	0.62	5.83
7 sites	1	0.44	0.85	3.44	1	0.40	0.77	2.50
	10	0.35	0.68	3.42	10	0.34	0.64	2.34
	100	0.34	1.01	9.41	100	0.32	0.62	6.06
11 sites	1	0.43	0.82	3.74	1	0.40	0.76	2.56
	10	0.36	0.72	3.62	10	0.34	0.65	2.24
	100	0.36	**1.05**	9.50	100	0.32	0.61	**5.80**

(a) – Rados

	Mean **writing** time (seconds)				Mean **reading** time (seconds)			
	Number \ Size	256 KB	1 MB	10 MB	Number \ Size	256 KB	1 MB	10 MB
1 site	1	0.36	0.72	1.74	1	0.34	0.64	1.78
	10	0.21	0.53	1.89	10	0.16	0.46	1.81
	100	0.46	1.26	9.75	100	0.45	1.10	8.85
7 sites	1	0.36	0.67	1.92	1	0.30	0.56	1.75
	10	0.22	0.56	2.11	10	0.18	0.42	1.67
	100	0.56	1.28	9.97	100	0.38	0.96	8.89
11 sites	1	0.38	0.67	1.91	1	0.31	0.62	1.80
	10	0.21	0.57	2.06	10	0.17	0.43	1.70
	100	0.55	1.32	9.76	100	0.40	0.97	**11.75**

(b) – Cassandra

	Mean **writing** time (seconds)				Mean **reading** time (seconds)			
	Number \ Size	256 KB	1 MB	10 MB	Number \ Size	256 KB	1 MB	10 MB
1 site	1	0.42	0.69	1.69	1	0.23	0.26	0.57
	10	0.24	0.34	1.81	10	0.14	0.25	0.50
	100	0.35	1.23	**12.20**	100	0.22	0.61	**3.95**
7 sites	1	0.41	0.62	1.69	1	0.22	0.36	0.59
	10	0.22	0.43	1.85	10	0.18	0.32	0.51
	100	0.40	1.32	11.54	100	0.25	0.66	3.94
11 sites	1	0.41	0.65	1.65	1	0.26	0.37	0.64
	10	0.24	0.33	1.93	10	0.19	0.26	0.49
	100	0.35	1.16	**11.86**	100	0.22	0.61	**3.93**

(c) – IPFS

Table 2: **Mean time** (seconds) to write or read one object with Rados (a), Cassandra (b) and IPFS (c) using 1, 7 and 11 sites. Bold values are particularly discussed in the text.

size of the object: the larger the object, the longer is the access time. Finally, we can observe that accessing to a large number of objects in parallel on each client degrades the performance (0.49 vs 3.93 seconds for the workloads $10 \times 10\,\text{MB}$ and $10 \times 100\,\text{MB}$ in reading with 11 sites). This trend is due to the load on each client that increases to perform the different computations. In addition to determining the hash, Rados and Cassandra clients should compute the object locations. For IPFS, we noticed that the client can only perform two requests simultaneously. This means that even if the YCSB module we implemented, generates 100 requests in parallel (using 100 threads), the IPFS client can only handle them two by two. This weak parallelism leads to worse performance.

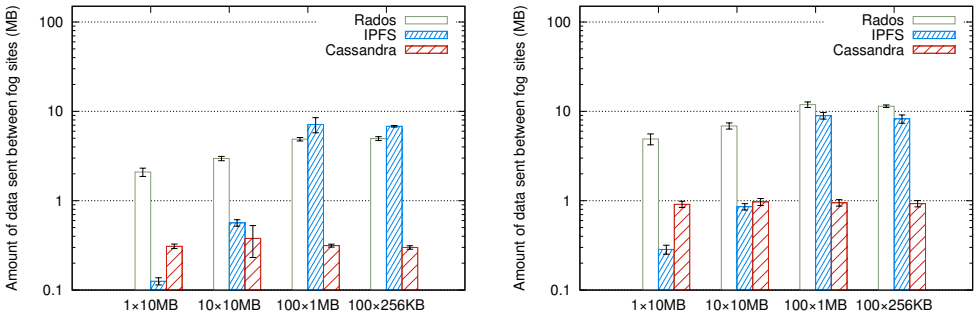
To summarize, Rados and Cassandra have stable access times, and only the load on each client seems to penalize the performance. For IPFS, additional experiments should be performed to better quantify the Kademlia DHT impacts on the access times. This is done in Section 5.

Inter-sites traffic analysis Figures 8 and 9 show the amount of network traffic sent between sites for the 7 and 11 sites scenarios. First, it is noteworthy that the traffic quantity exchanged between sites is relatively small in comparison to the amount of data stored (less than 2%). Even if this amount of inter-sites network traffic is low, it could have an important impact on the access times.

For Rados, the amount of traffic that is exchanged between sites is similar for writing and reading and depends on the number of object manipulated and not their size (by adding sites, we increase the number of clients and thus the total number of objects manipulated). As previously described in Figure 3(c), storage nodes send status and statistics to monitors that increases the amount of network traffic. As discussed in Section 3.1, it is possible to reduce the number of monitors (we used one monitor per site). While it will decrease the network overheads related to monitors, it will increase the traffic related to the OSD status reporting and clustermap retrievals.

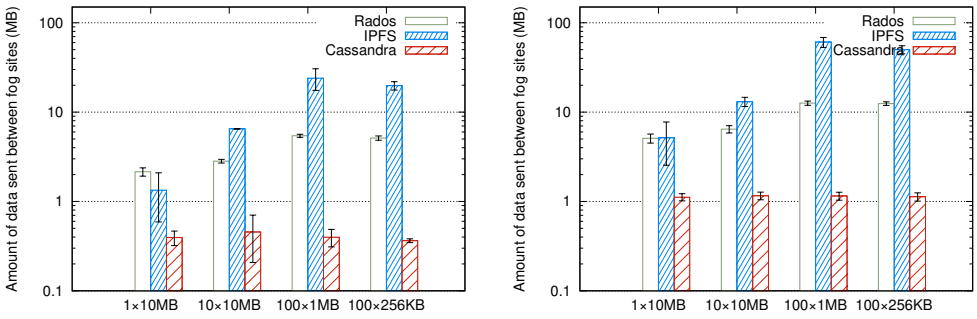
For IPFS, the inter-sites traffic corresponds to the Kademlia DHT messages that are used to determine and update object locations. Because the DHT is distributed among the sites, these operations generate network traffic. More traffic is exchanged for read accesses because in this case, the DHT is accessed twice: one time to retrieve the object location and a second time to announce the availability of the replica created on the server. We will see in Section 5 that this network traffic can be reduced for IPFS.

For Cassandra, the amount of traffic increases in a linear way with the number of nodes because each second every node sends a gossip packet to another one, as explained in Section 3.2. Because some traffics can be asynchronous and do not impact the access times, we computed the correlations between the amount of network traffic sent and the access times for read operations on the 7 sites scenario: 0.13 for Rados, -0.46 for Cassandra and 0.98 for IPFS. It confirms for Rados and Cassandra, the network traffic does not impact the access times. For IPFS, the correlation is higher because nodes have to wait an answer from the DHT before relocating the object and sending it to the client.



(a) – 7 sites (b) – 11 sites

Fig. 8: Cumulated amount of network traffic exchanged between all the sites while clients **write** objects on their sites. The scale is logarithmic and the error bar represents the standard deviation.

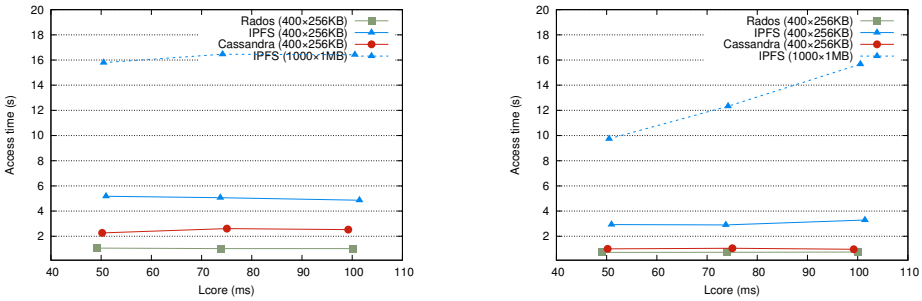


(a) – 7 sites (b) – 11 sites

Fig. 9: Cumulated amount of network traffic exchanged between all the sites while clients **read** objects located on their sites. The scale is logarithmic and the error bar represents the standard deviation.

To conclude, only Cassandra has a good behaviour in term of network traffic exchanged during a local access. The amount of network traffic is indeed lower than 1.5 MB using 11 sites. Details about all network traffic (intra and inter-sites) are given in appendix A.

Impact of inter-sites latency (L_{Core}) In this experiment, we want to determine how the network latency between the different sites impacts the object access times. In particular, we want to quantify whether accessing the Kademlia DHT for IPFS is critical. The value of L_{Fog} is set to 10 ms (like in the previous experiment) and the value of L_{Core} has been successively defined to 50 ms, 75 ms and 100 ms. We executed the 7 sites scenario but instead of manipulating 100 objects we increased this number up to 400. The goal was to generate a sufficient



(a) – **Mean writing time** (second) as a function of the L_{Core} latency

(b) – **Mean reading time** (second) as a function of the L_{Core} latency

Fig. 10: Mean access time to **write** and **read** one object when the value of L_{Core} latency vary from 50 ms to 100 ms A workload of 400 × 256 KB is used on 7 sites. For IPFS a workload of 1000 × 1 MB is also used.

number of requests to the DHT. Figure 10 shows the average time for writing and reading one object depending on the L_{Core} value. Regarding write accesses, the curves show that increasing the latency between Fog sites does not impact the performance. This is because there is no synchronous exchange for write operations as discussed in the previous paragraphs. Regarding read operations, we can observe for IPFS the penalty of accessing the DHT for a large number of objects (the curve increases from 9.74 s to 15.67 s according to L_{Core}). We also experimented with L_{Core} greater than 100 ms but Rados stopped to work because it was unable to perform the Paxos election.

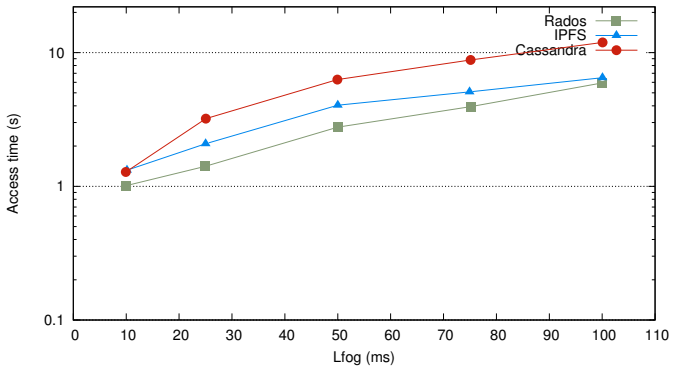


Fig. 11: Access time to write and read one object when the value of L_{fog} latency vary from 10 ms to 100 ms. Values in writing and reading are the same. A workload of 100 × 1 MB and a topology composed of 7 sites are used. Scale is semi-logarithmic on the y-axis.

Impact latency access to the Fog (L_{Fog}) Similarly to L_{Core}, we wanted to evaluate the impact of the L_{Fog} latency (the latency between clients located at the Edge and the servers in the site to which they belong). Clients can use wireless links that can be shared among a lot of users with packet loss and congestion, increasing the latency. The value of L_{Core} has been set to 50 ms while increasing the value of L_{Fog} from 2 ms to 100 ms. The latency between the servers of one site has not been modified (0.5 ms).

Figure 11 shows the average access time to write one object for Rados, Cassandra and IPFS as a function of the L_{Fog} value (trends being similar in writing and reading, we only draw the results for the write operation). Curves show that the latency L_{Fog} has an impact for the three systems. However, we expected that the impact of L_{Fog} would be the same for the three systems because for writing, the client just sends the data to a storage server of the site it belongs to. It appears that L_{Fog} latency has a bigger impact on Cassandra. Its access times are increased from 1.28 to 11.91 s (×9.30) whereas with Rados, access times are only increased from 1.01 to 5.02 s (×4.97) Figure 12 depicts this protocol: when a client writes an object, it first selects a server (with CRUSH for Rados, in a round-robin way for Cassandra and randomly for IPFS) and sends the object. All traffic behind this exchange is not governed by the L_{Fog} metric. To understand why the impact of Cassandra is more important than for Rados and IPFS. Indeed, the only extra traffic in writing for Cassandra is the forward that is performed between the servers of the site and thus, that is not impacted by the L_{Fog} latency. We also tried to use the "TokenAware" policy of Cassandra avoiding the forwarding mechanism but the result is not convincing. Further experiments are mandatory but left for future work.

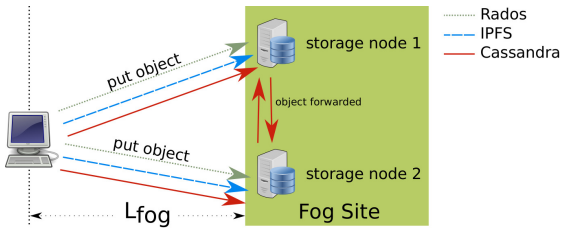


Fig. 12: Traffic sent by the client when writing for the three systems. The forward for Cassandra is not impacted by L_{fog} latency.

Summary These experiments evaluated the behaviour of the systems while manipulating objects stored locally (*i.e.*, on the closest site). Rados and IPFS get good performance in terms of access times but the amount of network traffic sent between the sites is linear to the number of objects accessed, which is a limiting factor. We observed the L_{Core} latency is not critical for the two systems although it should stay below 100 ms for Rados. For Cassandra, access times are

higher but the amount of network traffic sent between the sites depends only on the total number of storage nodes. This allows Cassandra to scale to a large number of accessed objects. However, Cassandra seems to be more impacted by high network latencies between the nodes than the other systems.

In the next part, we discuss experiments that write objects on one site and read them from another one.

4.3 Remote reading evaluation

The second experiment aims to evaluate the mobility criteria. Concretely, we want to analyze what are the impacts on the completion times (the time to read or write one object) when a client writes data on its local site and another client reads it from another location. We use the same topology as in the previous experiments with 7 sites. Rados, Cassandra and IPFS nodes are deployed in the same way but only two clients among the seven are used. Others sites provide nodes to IPFS DHT, monitors to Rados and generates gossip traffic for Cassandra.

Concretely, the following operations are performed: one client creates the objects on its site. Then, caches are dropped and another client located on another site reads the objects. Read is performed twice in order to analyze the benefits of having the implicit creation of a copy on the local site for IPFS. We remind that for Rados and Cassandra, objects are not explicitly relocated on the site the user performs read (*i.e.* data placement constraints are not modified dynamically in Rados and in Cassandra). We just evaluate the time to perform a remote read. The goal is to show the need for data relocation.

Tables 3(a), 3(b) and 3(c) show the access times we get in this scenario for Rados, Cassandra and IPFS respectively. Writing times are the same as in the previous experimentation when data are written on only one site and thus we have not reported the results in Table 3. Regarding read accesses, for the first ones, Rados client contacts directly the remote OSD storing the requested object (with a network latency equal to $L_{Fog} + L_{Core}$). So the increase of access time in reading is only due to the transfer of objects over the link having a L_{Core} latency between the client and the remote storage node. It takes 9.36 s to read 1 object of 10 MB remotely. This is roughly five times the time we measured in the previous experiment (2.53 s in Table 2(a)). We observe the remote read is more efficient with Rados with small objects. With a lot of objects, the parallelism between the objects limits the increase of access times due to the network latency. As an example with only 10 objects of 10 MB it is approximately 4 times longer to read remotely than locally (8.38 vs 2.27 s) but with 100 objects it becomes only 2.08 times longer to read remotely (12.14 vs 5.83 s).

With IPFS and Cassandra, requests are sent to a local storage node that locates the object and retrieves it before forwarding it to the client (as shown by the sequences diagrams in the Figures 5 and 6 on Pages 12 and 14 respectively). This mechanism increases the reading time. Moreover, only half of the requests (the ones that requested objects to the node which does not store them) was

Mean remote **reading** time (seconds) First read				Mean remote **reading** time (seconds) Second read			
Number \ Size	256 KB	1 MB	10 MB	Number \ Size	256 KB	1 MB	10 MB
1	1.80	3.34	**9.36**	1	1.84	3.42	9.21
10	1.80	3.31	**8.38**	10	1.83	3.29	8.10
100	1.72	3.09	**12.14**	100	1.72	3.09	11.66

(a) – Rados

Mean remote **reading** time (seconds) First read				Mean remote **reading** time (seconds) Second read			
Number \ Size	256 KB	1 MB	10 MB	Number \ Size	256 KB	1 MB	10 MB
1	1.46	3.01	**9.65**	1	1.41	3.06	9.43
10	1.53	3.97	12.43	10	1.53	3.75	12.52
100	3.77	8.26	**19.86**	100	3.87	8.14	21.43

(b) – Cassandra

Mean remote **reading** time (seconds) First read				Mean remote **reading** time (seconds) Second read			
Number \ Size	256 KB	1 MB	10 MB	Number \ Size	256 KB	1 MB	10 MB
1	0.99	1.24	3.03	1	0.18	0.43	**0.35**
10	0.70	1.15	5.23	10	0.16	0.36	0.31
100	1.28	**4.00**	38.85	100	0.21	0.61	2.96

(c) – IPFS

Table 3: **Mean time** (seconds) to read twice one object stored on a remote site with Rados (a), Cassandra (b) and IPFS (c). The topology is composed of 7 sites but only one is used to read. Bold values are particularly discussed in the text.

forwarded for local accesses. In this new experiment, because objects are stored remotely, the forward is performed for all requests.

For Cassandra, a remote read of $1 \times 10\,\text{MB}$ takes 9.65 s whereas it lasted only 1.78 s in the previous experiment to perform a local one (Table 2(b)). The metadata management does not imply an increasing of access time because with Cassandra, once the gossip has been propagated, every node can locate any object without any communication. Like in Rados, the increase of access time is only due to the object transfer using a high latency network link. We point out that Cassandra is the system that gets the highest increase of access times: a remote read of 100 objects of 10 MB takes 19.86 s per object whereas a local one in the previous experiment took 8.85 s.

For IPFS, when the read is done, the local node writes the object locally and updates the DHT asynchronously. The increase of access time for a remote read comes from the data transfer but also from the access to the DHT. In this scenario, each remote read requires a DHT request whereas some local read does not need it. We observe that access times for a remote read are 5 times more important than for a local access in all cases (as an example, 4.00 vs 1.23 s in Table 2(c) 100×1 MB), except for 100×10 MB.

The last columns of the Table 2(c) that, the access times for the second read in IPFS are in the same order of magnitude than in the previous experiment. Namely, IPFS spends 0.35 seconds to read 1 object of 10 MB (vs 0.57 s in the previous experiment). The small improvement of access times is due to the fact in this scenario, there is only one site sollicited while the DHT is spread on 7 sites. When a request is sent to a node that does not store the object, this node downloads it from all the nodes storing a replica. In a second read, the replica stored locally (due to the first read) and the original replica stored remotely are requested in parallel. This strategy may degrade the performance and increase the amount of traffic sent between the sites. To conclude on IPFS, access times are low because the requested nodes serve a copy of objects they kept in the first read.

For Rados and Cassandra, access times are identical for the first and the second read. The mobility of data is not explicit and thus the second access generates a remote read again. Because access times are high in case of remote reading, we next evaluate what is the influence of the network latency on these values.

Impact of inter-sites latency (L_{Core}) in case of remote access The goal of this experiment is to show how the remote read is impacted by L_{Core}, to show the need to relocate the objects on the local site. We performed the same test and varied the value of L_{Core} latency between 50 ms and 100 ms. But this time, we used the same scenario as in the second experiment, where a client reads data stored remotely.

Figure 13 shows the access times we got for reading. The increase of access times is as much important as in the case where L_{Fog} latency varied because data are sent using the inter-sites links. Rados and Cassandra behave like in Section 4.2, when the latency inside the sites were varying for a local access. As shown in the Figure 10 on the impact of L_{Core} latency in a local access scenario, metadata sent asynchronously does not impact the access times, thus only the object transfers are impacted by the network latencies. For IPFS, the increase of latency also impacts the DHT to determine the location of the objects. The access times for the second read are lower for IPFS, because as said previously, objects are retrieved from the local nodes which stored a replica at the end of the first read. The small increasing of access time is due to the DHT that needs to be accessed when the request for the second read is sent to the node that does not store the object. We suppose with 1000 objects, the second read will be influenced by L_{Core} value as for the local read in Figure 10: in this case, the DHT

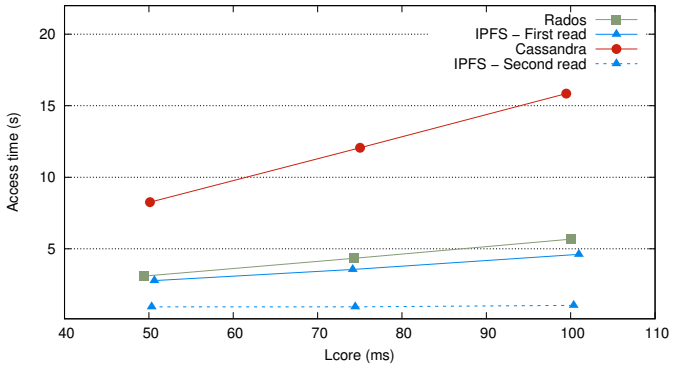


Fig. 13: Access time to read one object stored remotely when the value of L_{core} latency vary from 50 ms to 100 ms. A workload of 400×256 MB and a topology with 7 sites are used.

access will represent a large part of the access time. This additional experiment shows the latency L_{Core} has an important influence on access times for remote access. Objects should be relocated on the closest site instead of being reading remotely.

4.4 Summary of the evaluation of the three systems

To conclude, completion times for local accesses depend essentially on four parameters: the number of sites, the latencies inside and between the sites and the number of objects accessed. Table 4 summarizes for each system what are the parameters which have an impact on the local access times.

	Rados	Cassandra	IPFS
Number of sites	☹☹	☹☹	☺
Latency L_{fog}	☺	☹☹	☺☺
Latency L_{core}	☹☹	☺	☺
Number of accessed objects	☹	☺☺	☹☹

Table 4: Stability of the access times when the following parameters are increasing.

Rados is sensible to the latency between the sites (L_{Core}) because with a high latency greater than 100 ms, it stops to work. The Paxos protocol as well as the number of pools needed in a Fog context makes it non scalable to a huge number of sites and a huge number of objects increases the amount of network traffic but this traffic has a low impact on the access times.

Cassandra is sensible to the latency to reach the site of Fog (L_{Fog}) as well as the inter-sites latency (L_{Core}). Adding sites generates more network traffic and higher access times. Nevertheless, it differs from the others systems because the workload characteristics (the number of accessed objects and their size) has no influence on the access times. Cassandra is scalable to a very important number of objects but may be limited in number of sites (time to get a the status and the range of keys the other nodes are responsible for may become long).

We showed IPFS provides low access times that are not so impacted when the L_{Fog} and L_{Core} latencies are increased (as the number of objects access is small). The second experiment showed the relocation of data is needed in a Fog context: IPFS is the only system that is able to place automatically and natively the data as close as the user as possible. For this reason, IPFS may be considered to be used in a Fog environment. Nevertheless, the amount of network traffic exchanged between the sites, highly correlated to the access times, depends on the number of accessed objects which is a scalability challenge when a huge number of objects is accessed. In the next section, we evaluate more precisely the impacts of the DHT on the access times when IPFS is used with more clients. We also propose a solution to mitigate the network traffic exchanges in case of local reads.

5 Coupling IPFS with a Scale-Out NAS system

In this section, we focus on IPFS. We first evaluate the impact of the DHT on the access times and then, we propose to couple IPFS with a Scale-Out NAS to mitigate the use of the DHT for local reads. We evaluate the benefit of this approach regarding the cost of accessing remote meta data.

5.1 Cost of accessing a global DHT covering all the sites

We now evaluate in IPFS how the DHT impacts the access times. We focus on the local access of one site, when a client reads and write objects stored locally. These experiments are performed on a topology composed of 3 sites, each site containing 4 storage nodes. Only one site is associated to 10 clients in order to increase the number of objects accessed. The latency to reach the Fog site (L_{Fog}) is still equal to 10 ms but we increase the inter-sites latency (L_{Core}) to the extreme value of 200 ms. This value can correspond to the inter-sites latency of a mobile Fog site located in a train or in a bus (see Figure 1). We now use a `tmpfs` as a low-level backend to reduce the impact of the storage mechanisms on the access times and to conduct a more fair comparison between IPFS and IPFS coupled with a Scale-Out NAS (*i.e.*, we would like to remove the possible bias that can be generated by ext4 file system). Finally, we implemented our own benchmark by imitating the YCSB one. The goal was to be able to easily measure the access time of each object individually. Each of the 10 clients writes and reads 100 objects on the site, for a total of 1000 objects accessed. Figure 14 shows for a given client, the reading time of each object while accessing 100

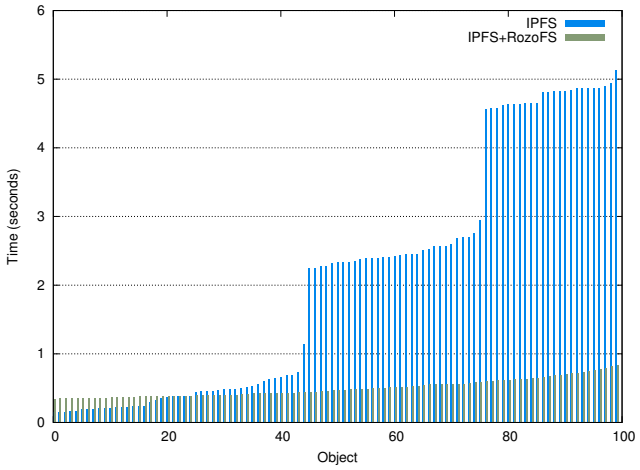


Fig. 14: Time to read every object for a given client and a given iteration with a workload of $100 \times 256\,\text{KB}$.

objects of 256 KB stored locally on the site. Values are given for IPFS used alone as well as for our solution labelled "IPFS+RozoFS" that will be detailed in the next section.

The first thing we observe when IPFS is used alone is the big gap in access times between the first object read and the last one. We also observe 4 plateaux in the curve, delimiting set of objects that have a similar reading time. The first plateau can be seen for the first 25 objects. These objects are read quickly because there are 4 IPFS nodes on the site and with 100 requests, 25 requests in probability will be sent to a node that stores the requested object. Therefore, the DHT is not accessed to locate these objects. Then, the second plateau (objects 25 to 43) shows the objects that are a bit longer to read than the previous ones. For these objects the DHT has to be accessed and the object relocated on the node the request is sent to. But the node storing the location of the object is located within the site, so there is no request sent outside the site, keeping a low access time. Finally, the longer objects to read (objects 43 to 100) are ones for which the location of the object is stored on a node located outside the site, reachable in one (objects 43 to 78) or several hops (objects 78 to 100).

This experiment shows that with a bigger latency between Fog sites and more clients, the DHT of IPFS increases significantly the access times. We now present how to fix this problem and how we obtain the result of the second histogram labelled "IPFS+RozoFS".

5.2 Coupling IPFS with a Scale-Out NAS

We propose to improve IPFS by adding a Scale-Out NAS deployed locally and independently on each site of Fog. Instead of storing the objects in the local

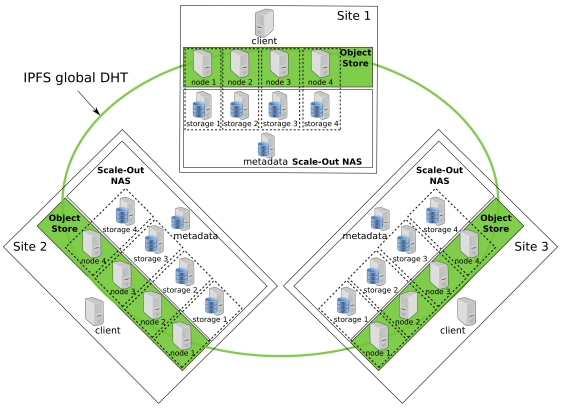


Fig. 15: Topology used to deploy an object store on top of a Scale-Out NAS local to each site.

storage device directly, IPFS nodes store the objects in the Scale-Out NAS deployed on the site. This enables all the nodes of a site to access all the objects stored on it without using the DHT to locate them.

In writing, clients send their objects to an IPFS node which stores them in the Scale-Out NAS system. The DHT is still updated asycnhronously to make the objects available for the other sites. In reading, when a locally stored object is requested, the IPFS node will find it in the Scale-Out NAS and thus will not request the DHT to locate it. The object will be sent directly to the client, lowering both the access times and the amount of network traffic exchanged between the sites of Fog. The DHT is only accessed to locate objects located on the other sites. We also point out the Scale-Out NAS system enables IPFS to work in case of network partitioning because clients can access objects stored locally without the need to contact the other sites to locate the objects.

We propose to evaluate this approach using IPFS on top a RozoFS [40], an open-source solution that is able to achieve high performance both for sequential and random access. RozoFS uses a metadata-server to locate data. It is not a problem to use such a metadata server because RozoFS cluster is limited to one site, IPFS nodes does not suffer from extra latency to reach it. RozoFS distributes (thanks to the Mojette erasure code) each object onto several storage nodes (here over 3 nodes). Just 2 nodes out of the 3 are necessary to decode and read the object. Any other Scale-Out NAS system could participate in theory to the demonstration of the interest of getting a storage backend under an Object Store system as IPFS. We compare the performance of our approach with the performance obtained using a traditional IPFS cluster.

The topology and the software architecture used is described in Figure 15. There are only 3 sites and each site contains one client, 4 storage nodes and a metadata server for RozoFS. The storage nodes of RozoFS are colocated on the IPFS nodes. The coupling may be easily implemented because as explained in

	Mean **writing** time (seconds)				Mean **reading** time (seconds)			
	Number \ Size	256 KB	1 MB	10 MB	Number \ Size	256 KB	1 MB	10 MB
3 sites	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

(a) – Using the default approach of IPFS.

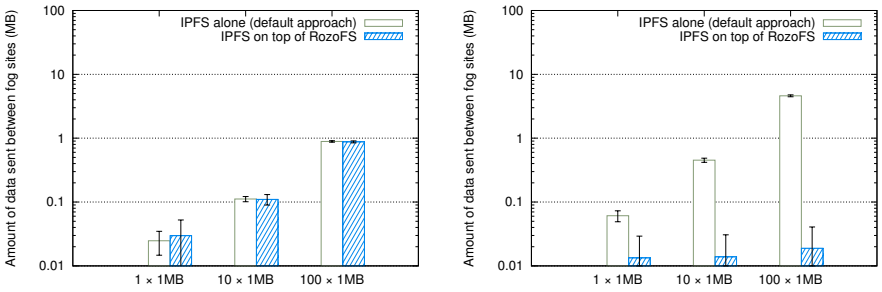
	Mean **writing** time (seconds)				Mean **reading** time (seconds)			
	Number \ Size	256 KB	1 MB	10 MB	Number \ Size	256 KB	1 MB	10 MB
3 sites	1	0.18	0.23	0.38	1	0.14	0.18	0.31
	10	0.17	0.22	0.43	10	0.14	0.18	0.36
	100	0.33	1.08	3.97	100	0.19	0.36	1.83

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

Table 5: **Mean time** (seconds) to write or read one object using IPFS alone (a) and IPFS on top of RozoFS (b).

the Figure 6 of the Section 3.3, an IPFS node does not access the DHT when it first finds the object locally. This coupling of IPFS with RozoFS consists to place the folder IPFS uses to store the objects in a RozoFS POSIX mountpoint. Nonetheless, we made a small modification in IPFS. By default, each time an IPFS node wants to read an object stored remotely, it accesses the DHT and contacts the nodes storing it. However, it does not request only this object but all the previous requested objects for which the download is not finished yet. This increases the network traffic between sites because objects are received several times. Our modification consists to request only the object to the nodes specified in the DHT.

Table 5 shows the access times when IPFS is used natively and on top of RozoFS with 3 clients writing and reading on their site (for a L_{Core} latency back 50 ms). The first thing we notice is the access times using IPFS alone are in the same order of magnitudes than the ones we got in the previous experiment (Table 2(c)). For example it takes 1.07 seconds to write one object in a 100×1 MB workload using 3 sites whereas it took 1.32 seconds in Table 2(c) using 7 sites. Only the scenario of 100×10 MB gets higher access times (11.54 seconds with 7 sites vs 3.92 with 3 sites). Using a `tmpfs` removes the need to flush data on a hard drive. The table also shows the writing access times are similar using RozoFS as a backend or IPFS alone. As an example, it takes 0.21 seconds to write one object with a 10×1 MB workload with IPFS used alone and 0.22 seconds when the coupling IPFS with RozoFS is used.Indeed in the both cases, the DHT is updated asynchronously. From these results we can say that RozoFS does not add an overhead in terms of access time. In reading, not to access the DHT reduces the access times. As an example 1.83 seconds are needed to read



(a) – Write (b) – Read

Fig. 16: Cumulative amount of network traffic exchanged between all the sites while clients write and read objects located on their sites.

one object of 10 MB (in a 100×10 MB workload) using IPFSon top of RozoFS whereas 1.98 seconds are needed when IPFS is used alone. In average, reading time are 34% shorter when IPFS is used on top of RozoFS.

Figure 16 shows the amount of network traffic sent between the sites. We observe in writing an equivalent amount of traffic because in the two cases, the DHT is updated asynchronously (0.89 MB with 100 objects). But in reading, the amount of network traffic has been dramatically reduced when RozoFS is used as a backend of IPFS. With 100 objects, the amount of network traffic is reduced from 4.62 MB when IPFS is used alone to 0.02 MB when we use our coupling. The explanation is the DHT is not accessed anymore for local accesses and the only network traffic between the sites is to maintain the routing table of the DHT. Finally, we observe in Figure 14 using 10 clients and an important inter-sites latency ($L_{Core} = 200$ ms) that the coupling solution provides a low guarantee of access times because the time to access each object is approximately the same (approximately 0.50 s).

To conclude on this part, accessing the DHT increases the access times, even to read objects stored locally. To solve this problem, we proposed to add a Scale-Out NAS system on each site. This solution prevents the DHT to be accessed in a local read, reducing the reading times and the amount of network traffic exchanged between the sites. This preliminary result is encouraging and invite us to continue in this direction in the future.

6 Related works

An important problem caused by the need of mobility is how to locate the objects. We can find simple analogies between network and Fog storage. A host on a network identified by its IP (Internet Protocol) address corresponds to an object on a site identified by its key. Mobility support is a problematic studied in network. The following approaches can be classified in two categories: either a centralized server establishes the relation between an address and the location of the node or the address contains some information about the location of the

node. In this last case, the address has to change when the node moves. In a Fog storage the two approaches are difficult to use. The first one causes difficulties of scalability whereas the second one is not acceptable because it means that the key of an object changes according to the location of the object. The following paragraphs give some examples of these two approaches.

In a traditional IP network, IP address has two roles, the first role is to allow the packet to be routed in the correct network and to the correct machine. The second role is to identify each machine of the network. In case of mobility, the address of a user has to change in most of IP network because of this first usage for routing: the address describes the location of the user. In Mobile IP networks [28], a user keeps it IP address, no matter its geographical position. In the Fog, a user needs to keep a low access time to its data, no matter its geographical position. In a Mobile IP network, the IP of the user is announced from the base station it is connected to allowing routing mechanisms. A similar concept might be found for the Fog storage.

A similarity can also be found with the Locator/Identifier Separation Protocol (LISP) proposed by Farinacci et al. [21]. In this work, the authors propose to separate the two roles previously described. The IP address is only used an identifier and the location of the machine is determined thanks to a directory service. If we transpose this approach to the Fog: the key of an object could be the identifier of the object and a directory service is used to locate the object. The main problem of this approach is the scalability of the directory service. Contrary to LISP, Taroko [42], a protocol to route datagrams in a mobile ad-hoc network, knows the location of the destination node from its address. Transposing this approach to a fog storage system, means that the key of the object contains the location of the object. The drawback of this approach is the key of the object will change when data are moved from a site to another. Object's key, like address in Taroko will change in case of mobility.

We found similar concepts in 3GPP networks [1]. In these networks, a centralized home location register (HLR) is used to determine the base station each user is connected to. It is used to route the packets to the users. This centralized approach does not seem scalable to locate objects because it does not provide a disconnected mode. Also, when the user is not directly connected to the network of its operator (roaming), the HLR to use is found from the International Mobile Subscriber Identity (IMSI) provided by the user. If we transpose this strategy to the Fog, it means that the object identifier should contain the site the object is stored to or at least the site storing the location of the object. The approaches used in 3GPP transposed in the Fog context mean that the key of the object is used to determine the metadata server storing the location of the object. The drawback is the same as in 3GPP: you cannot change your operator without changing your IMSI number. In the Fog, it means we cannot change the metadata server used by an object without changing the key of this object. To summarize, the 3GPP approach mixes the both strategies: using a "centralized" metadata server and storing the location in the identifier. We can point out that the problematic of network containment was not addressed.

The following propositions are not specially focused on the Fog computing but it shows that network field faces with similar problems. Partial solutions have also been proposed for Cloudlets and Mobile Clouds. In Cloudlets [27], Jararweh *et al.* propose two approaches, a centralized approach where the system tracks the mobile devices and a decentralized approach where the mobile device is in charge of managing its movement. In a Fog context, it can be transposed in the fact the user store its own metadata and is responsible of its object's placement. The main drawback of this approach is that the data movement is only in charge of the user. Data cannot be moved by the system itself, for example to optimize the placement for energetic considerations because the clients will not found its objects anymore. It can be difficult to notify the client because it can be offline when data movement is performed. The main difference between Cloudlets and Mobile Cloud is a difference of scale. While Mobile Cloud considers large site with several servers, the Cloudlet approach considers a site as containing just a small server.

The previous approaches showed the most used technique are to specify the location of an object in its key (which is a problem in a mobility context) or to use a centralized metadata servers (which causes scalability problem). It also exists some P2P approaches to distribute this centralized metadata server. Clarke *et al.* [15] proposed in Freenet to look for resources using a preference list to direct the requests from one node to another. This epidemic approach is similar to some routing protocols such as PRoPHET [33], used more recently in a low connected environment. The drawback of a such protocol is that it cannot guarantee that an existing resource can be found. The same problem occurs in most of the non structured P2P network such as GNUtella [43].

Structured peer to peer network address this issue. A distributed hash table such as Kademlia DHT guarantee an existing key is found but it looses the locality of metadata. In the Pastry DHT [14], node's locality is exploited by placing close nodes at near locations in the DHT. In the Fog context, it could be used to avoid sending requests outside the site when requests are for objects stored locally. Recently, Wilkinson *et al.* [53] store the location of the objects in a blockchain. Vorik *et al.* [50] have a similar approach, users store the files of other users in exchange of virtual money. A blockchain keeps trace of what is stored on each node. Because the blockchain is replicated on every node in the network, it allows clients in a Fog context, to locate any object and to access data in case of network partitioning. But this approach have also some drawbacks, especially in case of mobility. The blockchain is an append-only data structure, data can only be added, not modified and not removed. Thus, if the location of an object is added each time it is moved, the size of the blockchain will increase a lot. Also, the time to complete a transaction can be a problem: if a client moves, it does not want the location of the data be updated several minutes later: the blockchain may not be scalable beyond 7 transactions per second [17]. Brand *et al.* [10] use a totally different approach by proposing different sites to use different storage system. It also uses a hierarchical namespace to avoid updating the metadata for each new object stored.

To summarize this part, the approaches for mobility support do not meet the property of data locality and network containment (there is no locality in a naive DHT) or do not work in case of network partitioning (if the address of the metadata server is found in the object key, the metadata server has to be on the same partition as the client). The coupling between an object store solution and a Scale-Out NASsystem solves this problem in an original way.

7 Conclusion and Future work

We presented a list of expected properties for Fog storage systems and evaluated three off-the-shelf object store solutions (namely Rados, Cassandra and IPFS) using the Yahoo Cloud System Benchmark (YCSB). Performance is measured in terms of access times and network traffic. We also evaluated the impact on the access times of the network latencies between the sites of Fog and also between the clients and their closest site.

The first experiments concerned the evaluation of the performance in a local access as well as in a remote access scenario. It showed that IPFS is the best candidate for the Fog context because of its ability to scale to a huge number of site and the low access times it provided. We also showed that IPFS is less sensible to the network latency between the clients and the sites of Fog (contrary to Cassandra). The major drawback of IPFS is the need to access the DHT each time an object read is not stored on the requested node. That leads to generate a huge amount of network traffic between the sites and to an increasing of access times while accessing object stored locally.

To deal with this problem, we proposed to add a Scale-Out NAS system on each site. The Scale-Out NAS shares the object stored on a site among all the IPFS nodes of the site and avoids to access the DHT when an object to read is locally stored. Experiments using RozoFS as a Scale-Out NAS showed that it reduces the local reading times by 34% in average and the amount of network traffic between the sites.

Even if these first results are encouraging, many problems are still open. We considered the clients always contact their closest site. This is a strong assumption because some sites may provide more resources than others (in terms of storage space for example). A mechanism to determine the site to contact is needed or to forward asynchronously. Nodes churn and replication between the sites were not considered in this paper but they need to be considered in a more realistic Fog networking environment.

References

1. 3GPP: Network architecture. TS 23.002, 3rd Generation Partnership Project (3GPP) (Sep 2008)
2. Aazam, M., Huh, E.N.: Fog Computing and Smart Gateway Based Communication for Cloud of Things. In: Proceedings of the 2014 International Conference on Future Internet of Things and Cloud. pp. 464–470. FICLOUD'14, IEEE Computer Society, Washington, DC, USA (2014)

3. Abramova, V., Bernardino, J.: NoSQL databases: MongoDB vs Cassandra. In: Proceedings of the International C* Conference on Computer Science and Software Engineering. pp. 14–22. C3S2E '13, ACM, New York, NY, USA (2013)
4. Abramova, V., Bernardino, J., Furtado, P.: Evaluating Cassandra Scalability with YCSB, pp. 199–207. Springer International Publishing, Cham (2014)
5. Anwar, A., Cheng, Y., Gupta, A., Butt, A.R.: Mos: Workload-aware elasticity for cloud object stores. In: Proceedings of the 25th ACM International Symposium on High-Performance Parallel and Distributed Computing. pp. 177–188. HPDC '16, ACM, New York, NY, USA (2016)
6. Balouek, D., Carpen Amarie, A., Charrier, G., Desprez, F., Jeannot, E., Jeanvoine, E., Lebre, A., Margery, D., Niclausse, N., Nussbaum, L., Richard, O., Pérez, C., Quesnel, F., Rohr, C., Sarzyniec, L.: Adding Virtualization Capabilities to the Grid'5000 Testbed. In: Ivanov, I., Sinderen, M., Leymann, F., Shan, T. (eds.) Cloud Computing and Services Science, Communications in Computer and Information Science, vol. 367, pp. 3–20. Springer International Publishing (2013)
7. Benet, J.: IPFS - Content Addressed, Versioned, P2P File System. Tech. rep., Protocol Labs, Inc. (2014)
8. Bonomi, F., Milito, R., Natarajan, P., Zhu, J.: Fog Computing: A Platform for Internet of Things and Analytics, Studies in Computational Intelligence, vol. 546. Springer International Publishing (2014)
9. Bonomi, F., Milito, R., Zhu, J., Addepalli, S.: Fog Computing and Its Role in the Internet of Things. In: Proceedings of the First Edition of the MCC Workshop on Mobile Cloud Computing. pp. 13–16. MCC '12 (2012)
10. Brand, G.B., Lebre, A.: GBFS: Efficient data-sharing on hybrid platforms: Towards adding WAN-wide elasticity to DFSes. In: Computer Architecture and High Performance Computing Workshop (SBAC-PADW), 2014 International Symposium on. pp. 126–131 (Oct 2014)
11. Brewer, E.A.: Towards robust distributed systems (abstract). In: Proceedings of the Nineteenth Annual ACM Symposium on Principles of Distributed Computing. pp. 7–. PODC '00, ACM, New York, NY, USA (2000)
12. Byers, C.C., Wetterwald, P.: Fog Computing Distributing Data and Intelligence for Resiliency and Scale Necessary for IoT: The Internet of Things. Ubiquity 2015, 4:1–4:12 (Nov 2015)
13. Carns, P.H., Ligon L, W.B., Ross, R.B., Thakur, R.: PVFS: A Parallel File System for Linux Clusters. In: Proceedings of the 4th Annual Linux Showcase & Conference - Volume 4. p. 28. ALS'00, USENIX Association, Berkeley, CA, USA (2000)
14. Castro, M., Druschel, P., Hu, Y.C.: Exploiting network proximity in distributed hash tables. In: in International Workshop on Future Directions in Distributed Computing (FuDiCo. p. 52–55 (2002)
15. Clarke, I., Sandberg, O., Wiley, B., Hong, T.W.: Freenet: A Distributed Anonymous Information Storage and Retrieval System, pp. 46–66. Springer Berlin Heidelberg, Berlin, Heidelberg (2001)
16. Cooper, B.F., Silberstein, A., Tam, E., Ramakrishnan, R., Sears, R.: Benchmarking Cloud Serving Systems with YCSB. In: Proceedings of the 1st ACM Symposium on Cloud Computing. pp. 143–154. SoCC '10, ACM, New York, NY, USA (2010)
17. Croman, K., Decker, C., Eyal, I., Gencer, A.E., Juels, A., Kosba, A., Miller, A., Saxena, P., Shi, E., Gün Sirer, E., Song, D., Wattenhofer, R.: On Scaling Decentralized Blockchains, pp. 106–125. Springer Berlin Heidelberg (2016)
18. Dabek, F., Cox, R., Kaashoek, F., Morris, R.: Vivaldi: A decentralized network coordinate system. SIGCOMM Comput. Commun. Rev. 34(4), 15–26 (Aug 2004)

19. Donovan, S., Huizenga, G., Hutton, A., Ross, C., Petersen, M., Schwan, P.: Lustre: Building a file system for 1000-node clusters. In: Proceedings of the Linux Symposium (2003)
20. Dubey, H., Yang, J., Constant, N., Amiri, A.M., Yang, Q., Makodiya, K.: Fog Data: Enhancing Telehealth Big Data Through Fog Computing. In: Proceedings of the ASE BigData & SocialInformatics 2015. pp. 14:1–14:6. ASE BD&SI '15 (2015)
21. Farinacci, D., Fuller, V., Meyer, D., Lewis, D.: The Locator/ID Separation Protocol (LISP). Tech. Rep. Request for Comments 6830, Internet Engineering Task Force (Oct 2015)
22. Firdhous, M., Ghazali, O., Hassan, S.: Fog Computing: Will it be the Future of Cloud Computing? In: Third International Conference on Informatics & Applications, Kuala Terengganu, Malaysia. pp. 8–15 (2014)
23. Garcia Lopez, P., Montresor, A., Epema, D., Datta, A., Higashino, T., Iamnitchi, A., Barcellos, M., Felber, P., Riviere, E.: Edge-centric computing: Vision and challenges. SIGCOMM Comput. Commun. Rev. 45(5), 37–42 (Sep 2015)
24. Gettys, J., Nichols, K.: Bufferbloat: Dark buffers in the internet. Commun. ACM 55(1), 57–65 (Jan 2012)
25. Hong, K., Lillethun, D., Ramachandran, U., Ottenwälder, B., Koldehofe, B.: Mobile Fog: A Programming Model for Large-scale Applications on the Internet of Things. In: Proceedings of the Second ACM SIGCOMM Workshop on Mobile Cloud Computing. pp. 15–20. MCC '13, ACM, New York, NY, USA (2013)
26. Hupfeld, F., Cortes, T., Kolbeck, B., Stender, J., Focht, E., Hess, M., Malo, J., Marti, J., Cesario, E.: The XtreemFS architecture a case for object-based file systems in Grids. Concurrency and computation: Practice and experience 20(17), 2049–2060 (2008)
27. Jararweh, Y., Ababneh, F., Khreishah, A., Dosari, F., et al.: Scalable Cloudlet-based mobile computing model. Procedia Computer Science 34, 434–441 (2014)
28. Johnson, D.D.B., Arkko, J., Perkins, C.E.: Mobility Support in IPv6. Tech. Rep. Request for Comments 6275, Internet Engineering Task Force (Oct 2015)
29. Jorgensen, N.T.K., Rodriguez, I., Elling, J., Mogensen, P.: 3G Femto or 802.11g WiFi: Which Is the Best Indoor Data Solution Today? In: 2014 IEEE 80th Vehicular Technology Conference (VTC2014-Fall). pp. 1–5 (Sept 2014)
30. Lakshman, A., Malik, P.: Cassandra: A Decentralized Structured Storage System. SIGOPS Operating Systems Review 44(2), 35–40 (Apr 2010)
31. Lamport, L.: Paxos Made Simple, Fast, and Byzantine. In: Procedings of the 6th International Conference on Principles of Distributed Systems. OPODIS 2002, Reims, France, December 11-13, 2002. pp. 7–9 (2002)
32. Legout, A., Urvoy-Keller, G., Michiardi, P.: Understanding BitTorrent: An Experimental Perspective. Tech. rep., INRIA, Institut Eurecom (2005)
33. Lindgren, A., Doria, A., Davies, E.B., Grasic, S.: Probabilistic Routing Protocol for Intermittently Connected Networks. Tech. Rep. Request for Comments 6693, Internet Engineering Task Force (Oct 2015)
34. Markopoulou, A., Tobagi, F., Karam, M.: Loss and delay measurements of Internet backbones. Computer Communications 29(10), 1590 – 1604 (2006), Monitoring and Measurements of IP Networks
35. Maymounkov, P., Mazieres, D.: Kademlia: A Peer-to-Peer information system based on the XOR metric. In: International Workshop on Peer-to-Peer Systems. pp. 53–65. Springer (2002)
36. Nielsen, J.: Nielsens law of internet bandwidth (1998)

37. Padhye, J., Firoiu, V., Towsley, D., Kurose, J.: Modeling TCP Throughput: A Simple Model and Its Empirical Validation. SIGCOMM Comput. Commun. Rev. 28(4), 303–314 (Oct 1998)
38. Palankar, M.R., Iamnitchi, A., Ripeanu, M., Garfinkel, S.: Amazon S3 for Science Grids: A Viable Solution? In: Proceedings of the 2008 International Workshop on Data-aware Distributed Computing. pp. 55–64. DADC '08, ACM, New York, NY, USA (2008)
39. Pallis, G., Vakali, A.: Insight and Perspectives for Content Delivery Networks. Commun. ACM 49(1), 101–106 (Jan 2006)
40. Pertin, D., David, S., Evenou, P., Parrein, B., Normand, N.: Distributed File System based on Erasure Coding for I/O Intensive Applications. In: 4th International Conference on Cloud Computing and Service Science (CLOSER). Barcelone, Spain (Apr 2014)
41. Qiu, D., Srikant, R.: Modeling and performance analysis of bittorrent-like peer-to-peer networks. SIGCOMM Comput. Commun. Rev. 34(4), 367–378 (Aug 2004)
42. Ridoux, J., Kassar, M., Boc, M., Fladenmuller, A., Viniotis, Y.: Performance of Taroko: A Cluster-based Addressing and Routing Scheme for Self-organized Networks. In: Proceedings of the 2006 International Conference on Wireless Communications and Mobile Computing. pp. 109–114. IWCMC '06, ACM, New York, NY, USA (2006)
43. Ripeanu, M.: Peer-to-Peer architecture case study: Gnutella network. In: Peer-to-Peer Computing, 2001. Proceedings. First International Conference on. pp. 99–100 (Aug 2001)
44. Shvachko, K., Kuang, H., Radia, S., Chansler, R.: The Hadoop Distributed File System. In: Proceedings of the 2010 IEEE 26th Symposium on Mass Storage Systems and Technologies (MSST). pp. 1–10. MSST '10, IEEE Computer Society, Washington, DC, USA (2010)
45. Souza Couto, R.D., Secci, S., Mitre Campista, M.E., Maciel Kosmalski Costa, L.H.: Network design requirements for disaster resilience in IaaS Clouds. IEEE Communications Magazine 52(10), 52–58 (October 2014)
46. Sui, K., Zhou, M., Liu, D., Ma, M., Pei, D., Zhao, Y., Li, Z., Moscibroda, T.: Characterizing and improving wifi latency in large-scale operational networks. In: Proceedings of the 14th Annual International Conference on Mobile Systems, Applications, and Services. pp. 347–360. MobiSys '16, ACM, New York, NY, USA (2016)
47. Tang, B., Chen, Z., Hefferman, G., Wei, T., He, H., Yang, Q.: A Hierarchical Distributed Fog Computing Architecture for Big Data Analysis in Smart Cities. In: Proceedings of the ASE BigData & SocialInformatics 2015. pp. 28:1–28:6. ASE BD&SI '15, ACM, New York, NY, USA (2015)
48. Tatebe, O., Hiraga, K., Soda, N.: Gfarm grid file system. New Generation Computing 28(3), 257–275 (2010)
49. Vaquero, L.M., Rodero-Merino, L.: Finding Your Way in the Fog: Towards a Comprehensive Definition of Fog Computing. SIGCOMM Computer Communication Review 44(5), 27–32 (Oct 2014)
50. Vorick, D., Champine, L.: Sia: Simple decentralized storage. Tech. rep., Nebulous-Labs, Boston (2014)
51. Weil, S.A., Brandt, S.A., Miller, E.L., Maltzahn, C.: CRUSH: Controlled, Scalable, Decentralized Placement of Replicated Data. In: Proceedings of the 2006 ACM/IEEE Conference on Supercomputing. SC '06 (2006)

52. Weil, S.A., Leung, A.W., Brandt, S.A., Maltzahn, C.: 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. pp. 35–44. PDSW '07 (2007)
53. Wilkinson, S., Boshevski, T., Brandoff, J., Buterin, V.: Storj A Peer-to-Peer cloud storage network. Tech. rep., Storj Labs Inc. (2014)
54. Yannuzzi, M., Milito, R., Serral-Gracia, R., Montero, D., Nemirovsky, M.: Key ingredients in an IoT recipe: Fog Computing, Cloud computing, and more Fog Computing. In: Computer Aided Modeling and Design of Communication Links and Networks (CAMAD), 2014 IEEE 19th International Workshop on. pp. 325–329 (Dec 2014)
55. Yi, S., Hao, Z., Qin, Z., Li, Q.: Fog computing: Platform and applications. In: Proceedings of the 2015 Third IEEE Workshop on Hot Topics in Web Systems and Technologies (HotWeb). pp. 73–78. HOTWEB '15, IEEE Computer Society, Washington, DC, USA (2015)
56. Yi, S., Li, C., Li, Q.: A Survey of Fog Computing: Concepts, Applications and Issues. In: Proceedings of the 2015 Workshop on Mobile Big Data. pp. 37–42. Mobidata '15, ACM, New York, NY, USA (2015)
57. Yi, S., Qin, Z., Li, Q.: Security and Privacy Issues of Fog Computing: A Survey, pp. 685–695. Springer International Publishing, Cham (2015)
58. Zao, J.K., Gan, T.T., You, C.K., Mndez, S.J.R., Chung, C.E., Wang, Y.T., Mullen, T., Jung, T.P.: Augmented brain computer interaction based on fog computing and linked data. In: Intelligent Environments (IE), 2014 International Conference on. pp. 374–377 (June 2014)
59. Zhang, B., Mor, N., Kolb, J., Chan, D.S., Goyal, N., Lutz, K., Allman, E., Wawrzynek, J., Lee, E., Kubiatowicz, J.: The Cloud is Not Enough: Saving IoT from the Cloud. In: Proceedings of the 7th USENIX Conference on Hot Topics in Cloud Computing. pp. 21–21. HotCloud'15, USENIX Association, Berkeley, CA, USA (2015)

A Details of the observed network traffic

Table 6 gives the details of the amount of network traffic exchanged between the sites in the scenario using the workload of 100 objects of 1 MB on 7 sites. Compared to the Figure 8 and 9 (page 21), this Figure shows the amount of network sent between the sites but also the local overhead sent inside each site.

For Rados, it shows that one site sends more data than the others. Indeed, the site 1 sends 10 times more data because it is where the elected monitor is. It could lead to different performance on different sites because all the sites do not have the same role. The traffic is not well-balanced between the sites and some sites can have more important access times. With Cassandra, all the sites have the same role. There is no site sending more data than the others. We observe the impact of the forwarding mechanism described in the Section 3.2: inside each site, approximately 150 MB are exchanged whereas only 100 MB are stored (overhead about 50%). This overhead will be increased if we use more than two storage nodes per site because fewer requests will be sent directly to the node storing the object. Finally, we also observe the forward mechanism during the reading operation with IPFS: the amount of network traffic inside each site has

an overhead of 50%. Table 6 shows for IPFS the forwarding mechanism we got for reading: 50% more network traffic is sent inside the sites in reading Indeed this overhead is produced, when the client sends the request to the node which does not store the object. To conclude on Table 6, we saw Rados has a site sending more traffic than the others while Cassandra and IPFS have an overhead of network traffic inside each site due to the forwarding mechanisms.

Amount of data sent while **writing** (KB)							
	Site 1	Site 2	Site 3	Site 4	Site 5	Site 6	Site 7
Site 1	**157924**	5	6	8	5	6	6
Site 2	7	**164507**	6	7	7	7	7
Site 3	7	7	**149175**	6	8	8	7
Site 4	10	9	8	**164313**	9	9	9
Site 5	10	9	10	10	**157726**	11	10
Site 6	10	10	11	11	12	**161713**	12
Site 7	12	12	12	10	12	13	**163908**

Amount of data sent while **reading** (KB)							
	Site 1	Site 2	Site 3	Site 4	Site 5	Site 6	Site 7
Site 1	**142719**	7	8	8	8	6	7
Site 2	8	**162345**	8	9	9	10	9
Site 3	10	10	**153008**	11	10	9	8
Site 4	13	12	11	**174439**	11	11	10
Site 5	12	11	13	12	**157729**	13	11
Site 6	12	13	12	13	14	**174799**	13
Site 7	14	14	12	13	13	14	**153007**

(a) – Cassandra

Amount of data sent while **writing** (KB)							
	Site 1	Site 2	Site 3	Site 4	Site 5	Site 6	Site 7
Site 1	108400	14	40	41	35	21	37
Site 2	**551**	108225	16	12	18	15	24
Site 3	**617**	18	108259	37	25	37	29
Site 4	**632**	46	27	108237	28	32	32
Site 5	**639**	28	33	33	108223	31	31
Site 6	**640**	26	38	42	35	108061	40
Site 7	**657**	45	46	46	38	44	108251

Amount of data sent while **reading** (KB)							
	Site 1	Site 2	Site 3	Site 4	Site 5	Site 6	Site 7
Site 1	108292	19	40	46	35	36	40
Site 2	**632**	108187	18	14	21	17	27
Site 3	**668**	21	108209	41	29	44	33
Site 4	**683**	50	31	108190	34	40	38
Site 5	**691**	33	39	39	108195	40	39
Site 6	**693**	31	46	50	44	108611	50
Site 7	**710**	55	56	55	49	56	108218

(b) – Rados

Amount of data sent while **writing** (KB)							
	Site 1	Site 2	Site 3	Site 4	Site 5	Site 6	Site 7
Site 1	**107418**	268	168	232	100	164	288
Site 2	268	**109655**	265	24	116	276	137
Site 3	178	253	**112605**	162	41	24	159
Site 4	235	23	167	**109538**	181	186	233
Site 5	140	114	41	168	**108142**	209	321
Site 6	164	282	206	206	211	**107658**	105
Site 7	299	134	167	243	331	104	**108627**

Amount of data sent while **reading** (KB)							
	Site 1	Site 2	Site 3	Site 4	Site 5	Site 6	Site 7
Site 1	**165595**	738	667	563	157	450	989
Site 2	722	**161474**	926	292	384	681	378
Site 3	656	963	**157808**	434	352	43	580
Site 4	561	248	452	**170178**	812	565	940
Site 5	143	402	290	788	**162622**	954	845
Site 6	449	686	214	602	941	**150680**	334
Site 7	1038	401	546	923	818	357	**177598**

(c) – IPFS

Table 6: Mean amount of data **in kilobytes** sent between the sites (source site in column and destination in row) for the scenario where **100 objects of 1 MB** are written and read on **7 sites** using Cassandra (a), Rados (b) and IPFS (c).