An Object Store Service for a Fog/Edge Computing Infrastructure based on IPFS and Scale-out NAS

Bastien Confais
CNRS, LS2N, UMR 6004,
Polytech Nantes,
rue Christian Pauc, BP 50609,
44306 Nantes Cedex 3, France

Adrien Lebre
INRIA, LS2N, UMR 6004,
Mines de Nantes,
4 Rue Alfred Kastler,
44300 Nantes, France

Benoît Parrein
Université de Nantes, LS2N, UMR 6004,
Polytech Nantes,
rue Christian Pauc, BP 50609,
44306 Nantes Cedex 3, France

***Abstract*—Fog and Edge Computing infrastructures have been proposed to address the latency issue of the current Cloud Computing platforms. While a couple of works illustrated the advantages of these infrastructures in particular for the Internet of Things (IoT) applications, elementary Cloud services that can take advantage of the geo-distribution of resources have not been proposed yet. In this paper, we propose a first-class object store service for Fog/Edge facilities. Our proposal is built with Scale-out Network Attached Storage systems (NAS) and IPFS, a BitTorrent-based object store spread throughout the Fog/Edge infrastructure. Without impacting the IPFS advantages particularly in terms of data mobility, the use of a Scale-out NAS on each site reduces the inter-site exchanges that are costly but mandatory for the metadata management in the original IPFS implementation. Several experiments conducted on Grid'5000 testbed are analyzed and confirmed, first, the benefit of using an object store service spread at the Edge and second, the importance of mitigating inter-site accesses. The paper concludes by giving a few directions to improve the performance and fault tolerance criteria of our Fog/Edge Object Store Service.**

I. Introduction

The Internet of Things (IoT) brings new constraints: a huge number of limited devices with a need of low latency computing. While largely adopted, the Cloud computing paradigm is not well suited because it relies on only a small number of datacenters located far away from users. The Fog computing paradigm has been proposed by Cisco to cope with this latency requirement [1]. The idea is to deploy dedicated servers in micro/nano datacenters geographically spread at the Edge of the network, that is, close to the end-users.

Figure 1 illustrates a Fog/Edge infrastructure. Each Fog site is composed of a small number of servers providing computation as well as storage resources to users. Users/IoT devices that are located in the Edge and in the Extreme Edge are connected to servers of the closest site, enabling low latency access. Devices located in the Edge contact directly the servers located in the Fog whereas those located in the Extreme Edge have to cross a local network before reaching the Fog. The one-way network latency to reach the Fog (noted L_{Fog}) is comprised between 10 and 100 ms (latency of a local wireless link [2]). The latency between the Fog sites (noted L_{Core}) is comprised between 50 and 100 ms (mean latency of a Wide Area Network link [3]). Finally, the latency to reach the Cloud (noted L_{Cloud}) is variable and unpredictable [4].

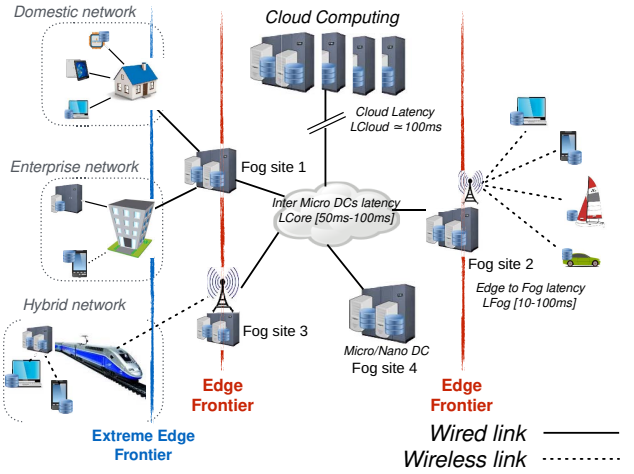


Fig. 1: Overview of a Cloud, Fog and Edge infrastructure.

Although using Fog/Edge facilities to deliver more efficient storage repositories has been discussed as early as 2012 [1], any solution has been proposed yet. We started to address this question in a preliminary study [5] with the ultimate goal of delivering a system similar to the Simple Storage Service (S3) of Amazon, a widely used service and a building block for hundreds of Cloud services. The main contributions of this study were:

- A list of properties a Fog Object Store Service should meet;
- An empirical analysis of three available storage systems, namely Rados [6], Cassandra [7] and InterPlanetary File System (IPFS) [8].

Among the three tested solutions IPFS filled most of the criteria expected for a Fog/Edge computing infrastructure. IPFS can be seen as an object store service built on top of the BitTorrent protocol [9] and the Kademlia DHT [10]. Both mechanisms are 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 (*i.e.*, the metadata).

The main issue of using a DHT for the metadata management in IPFS is that each time a user wants to access an

978-1-5090-3047-7/17 $31.00 © 2017 IEEE
DOI 10.1109/ICFEC.2017.13

object, it involves the DHT to locate the object if it is not available on the requested node. Because all the IPFS nodes are part of the Kademlia DHT, this increases the access time in addition to generating an important amount of network traffic between the sites, especially when a large number of objects is manipulated. In this paper, we propose to tackle this issue by deploying on each site a local Scale-out Network Attached Storage system (NAS). This system acts as a storage backend for the IPFS nodes belonging to the site. This way allows nodes to access any object stored locally on the site without using the global DHT. Nodes interact with the DHT only when the object they are looking for has been created on another site. Another solution may consists in adding a local metadata server on each site to determine if an object is stored locally on the site (and to determine the local node that stores it). Although this solution is feasible, it requires to deeply modify the code of IPFS and does not balance the load as well as the amount of data stored by the nodes within each site. Our approach preserves these aspects and only requires minor modifications in IPFS.

To validate our proposal, we implemented a Proof-of-Concept leveraging IPFS on top of RozoFS [11], and conducted several experiments on the Grid'5000 testbed [12]. We emulated three sites and measured the access times as well as the amount of network traffic exchanged between the sites for both local and remote conditions to validate ubiquitous access. The simultaneous observation of different Fog sites is another originality of this work. While the results are promising to deliver a well suited object store service for Fog/Edge infrastructures, we discuss current limitations of our changes and propose several directions to improve the performance while reducing as much as possible all the inter-site exchanges.

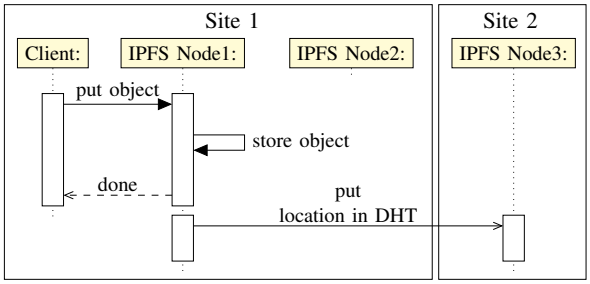
The remaining of the paper is organized as follows: Section II presents in details the limitation of using a DHT in IPFS for the metadata management. Section III gives an overview our solution that couples IPFS to a Scale-out NAS. Section IV deals with the experimental evaluations while Section V discusses two additional considerations related to our proposal. Related work are presented in Section VI. Finally, Section VII concludes this work and gives some perspectives.

II. IPFS: Overview and Limitation

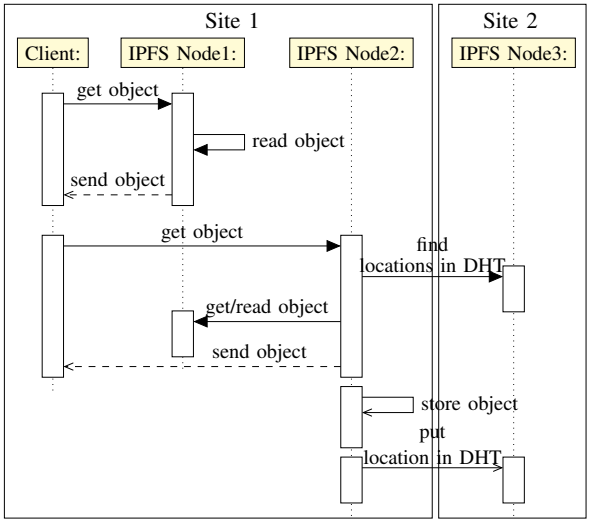
As briefly introduced, IPFS has been built on top of the BitTorrent protocol and the Kademlia DHT. The former enables the efficient relocation of objects between peers composing the infrastructure while the latter is used for the management of the metadata. The IPFS protocol is depicted in Figure 2. We underline that IPFS only enables the creation of immutable objects. Immutable objects make easier to maintain the consistency between all the replicas. Figure 2(a) depicts the creation of a new object: the client sends its object to any node on its nearest site. This node stores the object locally and `put` the location of the object in the DHT. Because the DHT does not provide locality, the node storing this metadata can be located in any node composing the Fog infrastructure. In our example, Node3 belonging to Site 2 stores the location of the object that has been created on Node1.

Figure 2(b) illustrates what happens when the client reads an object stored on its local site. Each time an IPFS node receives a request for a particular object, it first, checks whether this object is available on the node. In this case, the node sends the object directly to the client. Otherwise, the IPFS node should rely on the DHT protocol to locate the requested object. That is, it should *(i)* compute the hash based on the object id, *(ii)* contact the node in charge of the metadata, *(iii)* retrieve the object from the node(s) storing it (using the BitTorrent protocol), *(iv)* make a local copy while sending the data to the client, and *(v)* finally update the DHT in order to inform that there is a new replica of the object available on that node.

Figure 2(c) describes what does happen when an object is requested from another site (because the client moves from a site to another one or because the object is accessed by a remote client). In any case, the client contacts a node on the site it belongs to. Because the node does not store the object, the protocol is similar to the previous one involving the extra communication with the DHT. We underline that, thanks to the implicit replication performed by the BitTorrent protocol, if the client wants to read the object a second time on the same

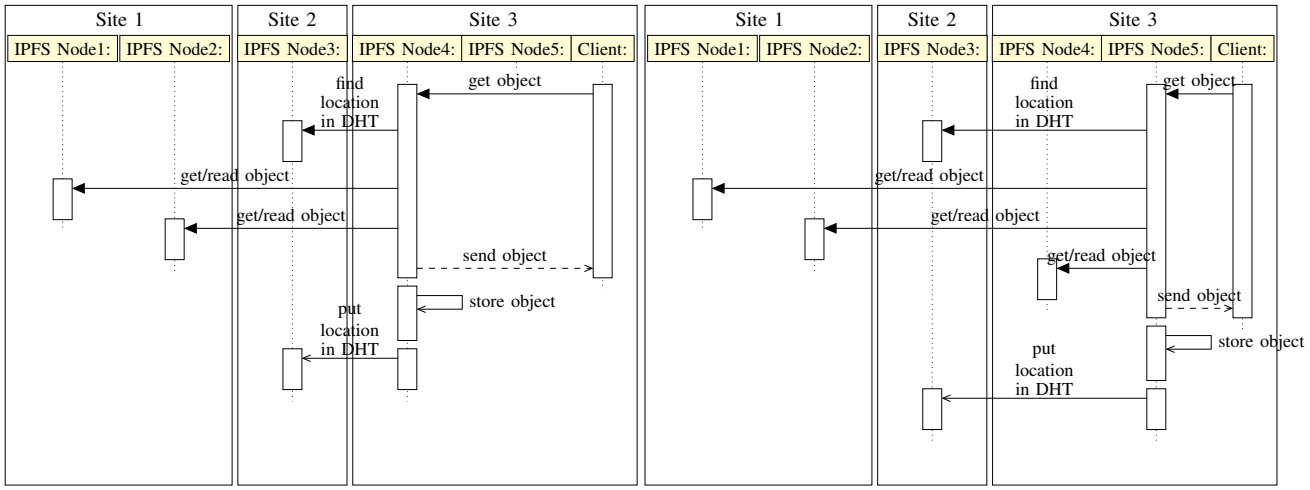


(a) – Write an object.



(b) – Read an object stored locally.

Fig. 2: Network exchanges when a Client writes or reads an object using **IPFS**.



(c) – Read an object stored remotely.

Fig. 2: Network exchanges when a Client writes or reads an object using **IPFS** (Cont).

node, the node will be able to satisfy the request without any interaction as described in Figure 2(b).

As observed in Section IV, IPFS provides better performance than an object store deployed in the Cloud computing. Nevertheless, it does not take advantage of the physical topology of the Fog infrastructure: When an object is created or available on one site due to BitTorrent copies, the other IPFS nodes that belong to this same site are not aware of the availability of the object and must rely on the DHT to locate the object. This is illustrated by the second get operation on Figure 2(c): Although the object is already available on Node4 of Site 3, Node5 contacts the DHT and retrieves the object from every node that owns a copy. Accessing the DHT to read an object stored locally on the site has two major drawbacks. First, it increases the access times as well as the amount of traffic exchanged between the sites and secondly, it prevents clients from accessing the objects stored locally in case of network partitioning (when a site cannot reach the others) because nodes of the DHT storing the location of local objects are unavailable. IPFS provides an efficient way to access objects remotely thanks to the BitTorrent protocol. But, it does not take the advantage of the Fog topology for local access. To tackle such a limitation, we propose to couple IPFS with a Scale-out NAS solution.

III. Coupling IPFS and Scale-out NAS Systems

To improve IPFS in a Fog Computing context, we propose to deploy a Scale-out NAS system independently on each site. Each system acts as the default storage backend for all IPFS nodes that belong to the same site as depicted in Figure 3. This approach allows each IPFS node of one site to access data stored by others nodes of the same site without any intrusive changes in the IPFS code as explained in the two next sections.

Solutions coupling an Object Store with a Scale-out NAS have already been proposed, for example using a Rados as a backend of a NFS service [13]. Nevertheless, our approach is

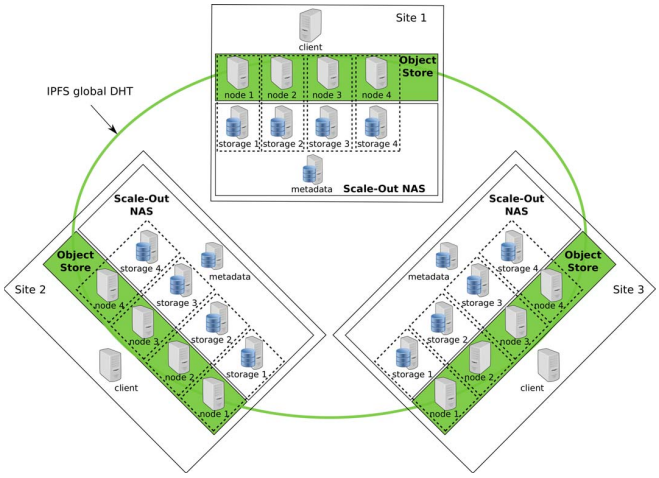


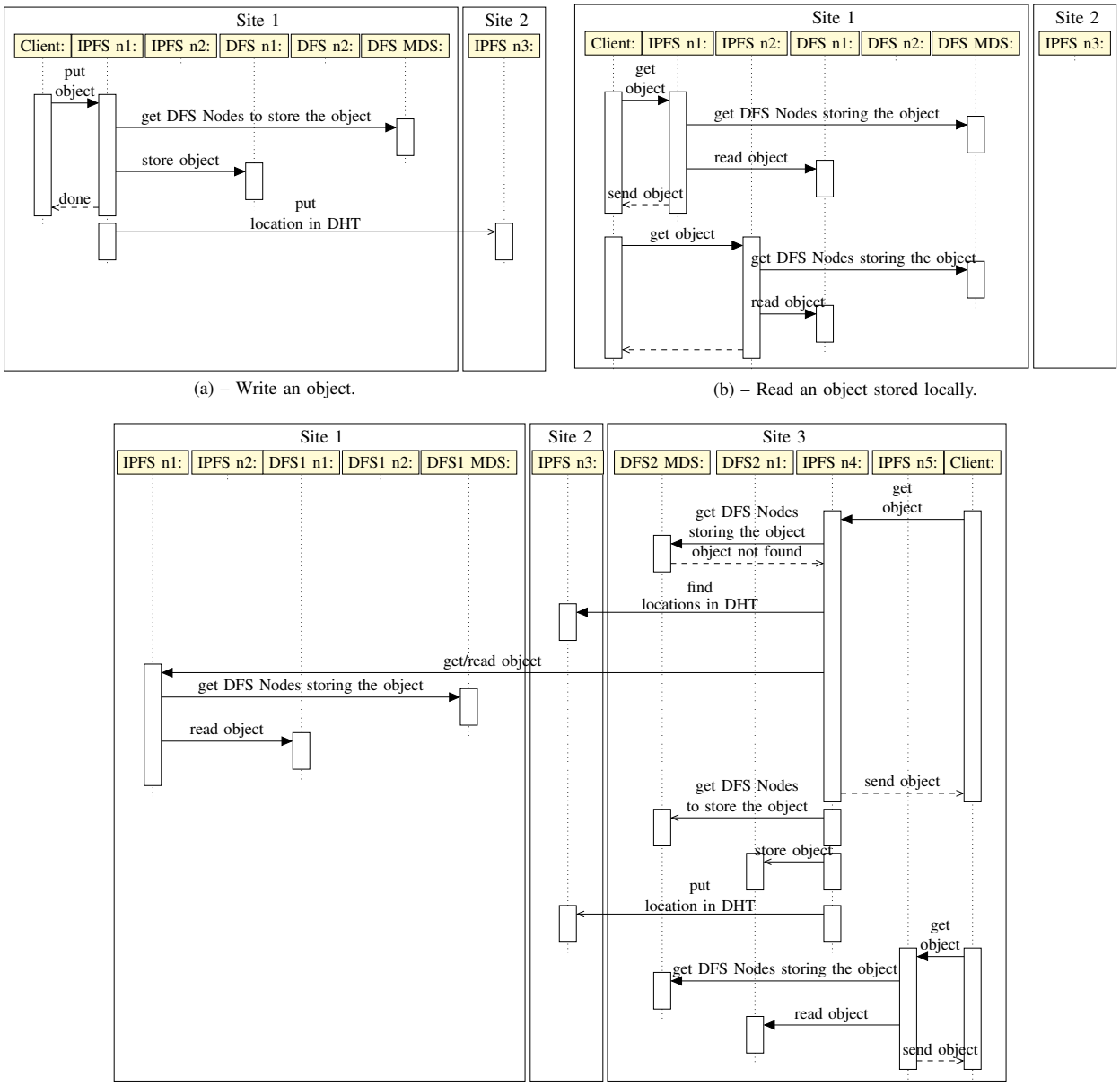
Fig. 3: Topology used to deploy an object store on top of a Scale-Out NAS local to each site.

different. In our topology, each site has its own Scale-out NAS and IPFS is the "glue" to create a global namespace among the different sites. A similar approach have been proposed in Group Based FileSystem (GBFS) [14] where different file systems at different locations are aggregated. However, this solution did not use an efficient protocol such as BitTorrent to share data accross the different locations.

A. Protocol Changes

The modifications in the protocol are illustrated in Figure 4.

In Figure 4(a), we can observe there is no change in the protocol for the creation of a new object. The difference appears when a client wants to access an object. Figure 4(b) depicts these changes: Any IPFS node on a site can see all the objects manipulated by the other nodes of the same site thanks to the Scale-out NAS system. As a consequence, regardless the



(a) – Write an object.

(b) – Read an object stored locally.

(c) – Read an object stored remotely.

Fig. 4: Sequence diagrams of the network traffics when a Client writes or reads an object using **IPFS on top of DFS** locally deployed on each site.

node on the local site the request is sent to, the requested node checks if the object exists in the Scale-out NAS and sends it to the client if it is available. Contrary to the original IPFS design, there is no need to access the DHT and to download the object from the other nodes located on the site.

Similarly, Figure 4(c) shows how the protocol behaves in case of remote access. When the client is connected to a remote site, the request is sent to any node. The node checks in the local Scale-out NAS backend if the object is found. If it does not find the object, it uses the DHT to determine its location. Then, it downloads the objects from a node located on the remote site, stores it locally, *i.e.*, in the shared backend and updates the DHT. Next accesses for this object from any node of this site will be satisfied without any extra-communication with the DHT.

To sum up, our approach limits the amount of network traffic sent in case of local reads: when a client reads an object available locally on the site, there is no need to access the DHT, regardless the node the request is sent to. In the other situations, the scenario as well as the amount of traffic

exchanged between the sites remains the same as with the default approach. We discuss in the next section, the evaluation we performed to quantify this gain.

B. IPFS Code Modifications

We performed some modifications in the source code of IPFS[1]. First, we moved the folder used by IPFS to store the objects on the Scale-out NAS mountpoint without moving the local database used by each node. Secondly, we removed the cache used by the blockstore module of IPFS. The cache is designed to keep in memory the fact an object exists or not. This modification is mandatory in order to enforce a node to check in the Scale-out NAS system before using the DHT to locate the requested object. Finally, we modified the way IPFS requests objects to other nodes. By default, each time an IPFS node wants to read an object stored remotely, it looks for the peer responsible for the object by using 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 unnecessarily increases the network traffic between sites. The modification consists in requesting only the object to the nodes specified in the DHT.

Finally, we disabled the replication of metadata in IPFS for all the software architectures we considered. This reduces the system resiliency if a node becomes unreachable (location may not be found even if replicas of the object are available) but it enabled us to measure the minimum amount of network traffic exchanged between the sites that is possible to have. We also removed the limit of the number of threads created by IPFS in order to prevent bottleneck in IPFS.

IV. Experimental evaluation

In this experimental evaluation, we compare the performance of *(i)* a Cloud based object-store service, *(ii)* the default IPFS solution and *(iii)* our approach coupling IPFS and Scale-out NAS systems. We underline the comparison with a Cloud based solution aims to illustrate the relevance of designing an object-store service that can benefit from the Fog/Edge specifics. The first paragraph presents our experimental protocol, the second deals with local accesses and finally, the third discusses results we gathered for remote access scenarios.

A. Material and Method

Our evaluations have been performed using three different software architectures:

1) IPFS in its default configuration deployed into a regular Cloud;
2) IPFS in its default configuration deployed across a Fog/Edge infrastructure;
3) IPFS coupled with independent Scale-out NAS solutions in a Fog/Edge context.

We selected RozoFS [11] as the Scale-out NAS solution. RozoFS is a POSIX compliant open-source solution that achieves good performance in throughput and I/O rate both for sequential and random access. Similarly to other Scale-out network file systems, it uses a metadata-server to locate the data. The distribution of the data across the I/O servers is realized by the Mojette erasure code that makes the system fault tolerant (2 Mojette projections out of 3 are necessary to reconstruct the data). More details are given in the article of Pertin *et al.* [11]. We emphasize this aspect because it leads to an overhead in writing (50% overhead in size *i.e.*, one redundant projection to combat one failure). This aspect is further discussed in Section V. The metadata server may also be replicated within the site in order to avoid any Single Point of Failure (SPOF) in the Scale-out NAS. We highlight that our proposal can be coupled with any kind of Scale-out network file systems such as GlusterFS [15], Lustre [16] or even Rados [6]. We chose RozoFS because we have a good knowledge of the internal mechanisms of the system, which helped us for the analysis of the results. Moreover, RozoFS achieves good performance regarding I/O access.

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

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

- local wireless link [2], $L_{Fog} = 10ms$, latency between clients and the fog site they belong to;
- wide area network link [3], $L_{Core} = 50ms$, latency between each fog site;
- and the latency to reach a cloud [17]; $L_{Cloud} = 100ms$.

Inside each Fog site and inside the Cloud, the latency between the servers has been set to $L_{site} = 0.5$ ms. Latencies were emulated using the Linux Traffic Control Utility (`tc`). We have not modified the throughput of the network links, *i.e.*, $10Gbps$ in our testbed for both intra and inter-sites links. According to Padhye *et al.* [18], the TCP throughput depends on packet loss and link latency. Measuring the throughput using *iperf* shows 2.84 Gbps for clients to reach a Fog node, 9.41 Gbps between the Fog nodes of a same site, and 533 Mbps between the sites. The measured throughput to reach the Cloud platform is 250 Mbps.

We performed experiments on two scenarios:

- a first scenario where one client per site writes objects on the site it belongs to and reads them;
- a second scenario where one client located on the first site writes objects and then, another client located on another site reads them.

[1] https://github.com/bconfais/go-ipfs/tree/common_backend_v1a

The first scenario is designed to evaluate the performance in terms of locality and isolation between the workloads whereas the second one enables us to quantify the cost of accessing objects from another location. For every write and read operation of each scenario, each client sends the request to an IPFS node selected randomly on the site it belongs to in order to balance the load among all the nodes of the site (for the cloud-based scenario, the selection is performed throughout the 12 nodes). In real-world applications, written objects are either never/rarely accessed or are accessed a lot of times [19]. This corresponds to a "zipfian" distribution [20]. The second read in the second scenario is an illustration of this pattern. We assume if a file is requested once, the file will be requested more than one time. We also consider the worst case in which all the objects written are requested in the reading phase.

Experiments have been performed in the spirit of reproducible research [21][2]. We measured access times using our own benchmark script and the amount of data exchanged between the sites using `iptables`. The access time corresponds to the time to complete the operation from the client point of view. We highlight that we previously performed experiments using the Yahoo Cloud System Benchmark [20] with an IPFS module we developed [5]. However, YCSB takes into account the time to randomly generate the written objects. This time may be an important part of the measured access time, especially when big objects of 10 MB are written and a `tmpfs` is used as low level back-end. We voluntary choose to develop our own benchmark in order to improve the quality of our analysis. Our benchmark is similar to YCSB and does not modify the order relations between the systems but reduced the writing time by a factor of two. Moreover, it enabled us to measure the time of each operation in a fine-grained manner. Each scenario has been run 10 times for consistency in results.

B. Write/Read Operations from the Same Location

In this section, we discuss the results obtained for the local access scenario, *i.e.*, when each client manipulates data simultaneously from their respective site. We also compare the access times we get in the case where objects are stored in a regular Cloud infrastructure.

1) Access Times: Table I shows the mean time to put or to get an object in each software architecture. The number of objects as well as their size correspond to what is written and read on each site in parallel.

Table I(a) presents the access times for the Cloud architecture. Each client (3 in our experiment) writes objects on their closest site and reads them. The clients work simultaneously. Results show significant access times with respect to the amount of data that is manipulated. Creating a single object of 256 KB takes 1.72 seconds and it requires up to 3.07 seconds in average for one object of 10 MB. If we increase the number of access in parallel, the maximum throughput that can be obtained for write operations is around 36 MB/sec ($100 \times 10/27.58$). Access times for read operations are better than the write ones for several reasons. First, some objects are retrieved without any indirection to the DHT (*i.e.*, when the IPFS request is sent to the IPFS node that stores the requested object). Secondly, the TCP throughput is higher for downloading than uploading with a maximum around 90 MB/sec ($100 \times 10/11.24$). This is due to the management of Linux TCP connections and the block granularity of IPFS for sending objects. In any case, these values are still important with respect to the amount of data that is manipulated. It confirms the impact of the latency on the TCP throughput [18].

Table I(b) presents the access times obtained when IPFS is deployed at the Fog level. Results clearly show the benefit of deploying the object store service on Fog/Edge resources. First, access times are significantly better with a maximum throughput per client around 500 MB/sec (4 Gbps). That is, half of the optimal performance for a 10 Gbps network interface (5 Gbps). Secondly, the impact between the clients is only related to the DHT traffic whereas in the Cloud architecture, all accesses are distributed across all nodes.

Tables I(c) shows the times to write and read objects using IPFS on top of RozoFS. First, we observe in the two Fog architectures, the writing times are in the same order of magnitude. It takes 3.92 seconds per object to write 100 objects of 10 MB with the default approach and 3.97 seconds using the couple IPFS with RozoFS. In other words, using a Scale-out NAS system such as RozoFS does not add an overhead. This is a valuable result because we can erroneously believe that using a complex software brick such as a Scale-out network file system can add some penalties. For reading, we observe the couple IPFS/RozoFS has better access times than the default approach (34% better in average). This is particularly visible for accessing a small number of small objects. As an example, it takes 0.14 seconds to read a 10 MB object whereas it requires 0.25 seconds with the original IPFS (almost twice). As previously described in Figure 4(b), the use of a Scale-out NAS enables IPFS nodes to directly satisfy the request wherever the object has been created on the site. The gain becomes smaller for large objects as the cost of accessing the DHT becomes less important regarding the time to send the object to the client. Moreover, as the probability to contact the node storing the object is $1/4$ in the original IPFS architecture, reading a large number of objects leads to a less significant gain (over 100 requests, 25 will be directly satisfied). With larger sites (*i.e.*, composed of a more important number of servers), the probability to contact the right node directly will decrease significantly and the gain of using a Scale-out system will be more visible.

2) Network Traffic: Figures 5(a) and 5(b) show the amount of network traffic exchanged between sites while writing and reading for the IPFS and IPFS/RozoFS solutions. We do not present the traffic for the Cloud-based approach as the traffic is equal to the amount of manipulated data. Moreover, we observe the amount of traffic sent between the sites depends only on the number of accessed objects and not on the object

[2]Our benchmark code as well as results are available at https://github.com/bconfais/benchmark

	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	1.72	2.14	3.07	1	1.47	1.88	3.04
	10	1.53	2.00	7.97	10	1.35	1.77	5.22
	100	2.29	5.55	27.58	100	1.57	2.62	11.24

(a) – Using a centralized Cloud infrastructure to store all the objects.

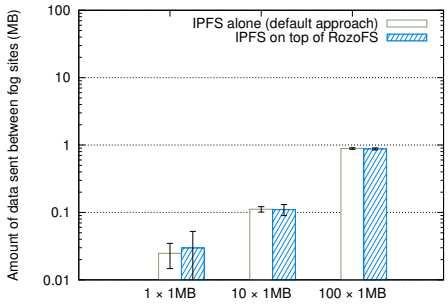
	Mean **writing** time (seconds)				Mean **reading** time (seconds)			
	Number \ Size	256 KB	1 MB	10 MB	Number \ Size	256 KB	1 MB	10 MB
3 sites	1	0.17	0.22	0.34	1	0.25	0.28	0.54
	10	0.17	0.21	0.40	10	0.26	0.27	0.54
	100	0.33	1.07	3.92	100	0.29	0.50	1.98

(b) – Using the default approach of IPFS.

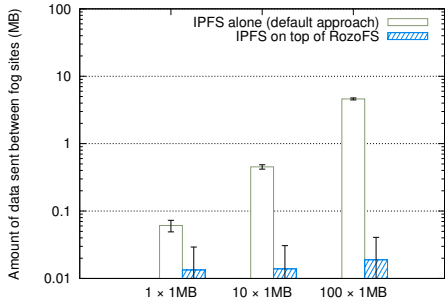
	Mean **writing** time (seconds)				Mean **reading** time (seconds)			
	Number \ Size	256 KB	1 MB	10 MB	Number \ Size	256 KB	1 MB	10 MB
3 sites	1	0.18	0.23	0.38	1	0.14	0.18	0.31
	10	0.17	0.22	0.43	10	0.14	0.18	0.36
	100	0.33	1.08	3.97	100	0.19	0.36	1.83

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

TABLE I: **Mean time** (seconds) to write or read one object under different conditions (the number on the left indicates the number of operations that are executed in parallel on each client).



(a) – Write



(b) – Read

Fig. 5: Cumulative amount of network traffic exchanged between all the sites while clients write and read objects located on their sites.

sizes. So, for a better readability, we arbitrary choose to present only the results for objects of 1 MB.

Figure 5(a) presents the value for write operations. As expected, the same amount of network traffic is sent using both approaches. In addition to the messages related to the DHT (heartbeats), the two approaches send the location of the written objects..

For reading, the traffic generated by the default approach is mainly the DHT requests: the more objects are manipulated, higher the traffic. In our approach using RozoFS, we only observe the traffic related to the management of the DHT. Thus according to the duration of the experiment and because the amount of data is really low, we can see some fluctuations monitored by the standard deviation.

To conclude, our approach using a Scale-out NAS is better than the two other approaches, both for access times and for the amount of traffic sent in reading. The Scale-out NAS removes the need to access the DHT when clients read objects already available somewhere on the site it is connected to.

C. Write from One Location, Read from Another One

In this section, we discuss the results obtained for the remote access scenario, *i.e.*, when one client creates data on its site and another client from another site access the created objects twice. Only read results are presented because writing locally has been already discussed in the previous section. Also, we do not consider the Cloud-based architecture in this scenario. Similarly to the results in writing, the read values will follow the same trend as to the one previously presented (there is no change because all clients are at the same distance from the Cloud).

1) Access Times: Tables II(a) and II(b) show the access times for the first and the second read operations.

In the first read, the times are in the same order of magnitude. This is due because the two approaches use the DHT to retrieve the location of the object and then copy the object on the second site. This trend corresponds to what we describe in Figures 2(c) and 4(c): the exchanges between the

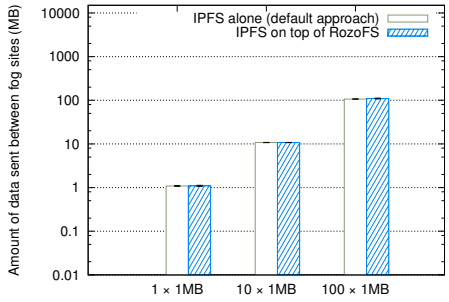
	Mean reading time (seconds) **First read**				Mean reading time (seconds) **Second read**			
	Number \ Size	256 KB	1 MB	10 MB	Number \ Size	256 KB	1 MB	10 MB
3 sites	1	1.39	1.92	13.07	1	1.01	1.85	3.70
	10	1.01	1.92	6.48	10	0.70	1.31	5.95
	100	0.94	2.02	9.76	100	0.71	1.37	6.08

(a) – Using the default approach of IPFS.

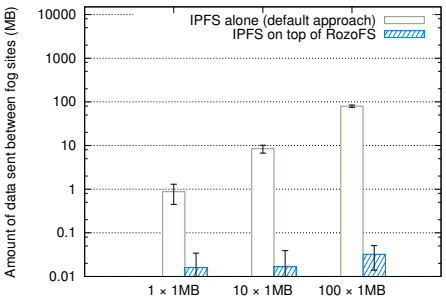
	Mean reading time (seconds) **First read**				Mean reading time (seconds) **Second read**			
	Number \ Size	256 KB	1 MB	10 MB	Number \ Size	256 KB	1 MB	10 MB
3 sites	1	1.35	3.86	13.21	1	0.15	0.19	0.31
	10	1.11	2.17	8.40	10	0.14	0.19	0.35
	100	1.09	2.51	9.22	100	0.33	0.46	1.86

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

TABLE II: **Mean time** (seconds) to read one object twice with IPFS using the default approach (a) and using a shared backend in each site (b).



(a) – First read



(b) – Second read

Fig. 6: Cumulative amount of network traffic exchanged between all the sites while clients read objects twice.

sites are the same. The requested node looks for the object location in the DHT and downloads the object from the node it is stored on. The only difference is the remote node retrieves the object from the Scale-out NAS before sending it to the node that requests it. The object is finally sent back to the client. According to the presented access times, this indirection through RozoFS on the remote site does not add an important overhead.

For the second read, we observe our approach gets better access times than the default one. As an example, with RozoFS it takes 1.86 seconds per object to read 100 objects of 10 MB whereas it takes 6.08 seconds in the default approach. Although the object is copied locally in both approaches, the probability to contact the right node for the second read (*i.e.*, the node that satisfied the first request) is $1/4$. If the client does not contact the right node, the process will be the same as for the first read. The only difference is the DHT will inform the node that it can retrieve the object from both the node on the remote site and from the node that previously retrieved and copied the object in the first read. In the IPFS/RozoFS approach, because of the shared back-end, objects are seen as stored locally by the requested node, the DHT is not accessed and nothing is downloaded from the remote site. As a consequence, access times are much faster. This behavior has been explained in Figure 2(c) and 4(c).

a) Network Traffic: Figures 6(a) and 6(b) show the cumulative amount of network traffic exchanged between the sites during the two reads.

For the first read, the same amount of data is exchanged between the sites in the two approaches. As previously explained, the objects must be downloaded from the site they were created. We highlight the traffic related to the DHT management is not significant enough in comparison to the amount of exchanged data to be visible.

For the second read, a lot of traffic is sent between the sites for the default IPFS approach. As explained in addition to the DHT request, the object is retrieved from the neighborhood and once again from the remote node where the object was created. Our IPFS/RozoFS approach enables the object store service to satisfy the requests completely by the local NAS. This way reduces the amount of traffic between sites only to the mandatory one. Once an object has been retrieved it is available for any node of the site. The default approach uses a local replica too but it is not shared among all the nodes of the site. Thus, nodes need to access the DHT to be able to retrieve it.

The main results of these experiments are:

- Performance of local writes are not impacted by the overhead of the Scale-out NAS system;

- Local reads are handled by the local Scale-out NAS system, preventing DHT accesses. Experiments showed an improvement of 34% of access times compared to a traditional IPFS cluster and avoid most of network traffic;
- These same improvements are shown in the second read of remote access scenario.

Although the IPFS/RozoFS Proof-of-Concept is promising, we discuss in the next section two additional considerations.

V. Additional Analysis

In this section, we discuss the possible side effects of the storage spaces induced by the fault tolerance of the underlying Scale-out NAS. We also discuss a performance limitation that may occur in some cases and propose to solve it by clustering IPFS nodes within a site into one virtualized IPFS node.

A. Storage Space Control

It is noteworthy that an approach based on a Scale-out NAS may add an overhead in terms of storage space according to the underlying fault tolerance policy (full replication, erasure-code etc.). When IPFS is used alone, the object is written on the IPFS node receiving it, without any replication. In our IPFS/RozoFS implementation, RozoFS writes a 50% overhead due to the Mojette erasure code [11] in order to support one failure at the NAS level. However, our proposal becomes more space efficient than the original IPFS for subsequent reads: in the default approach the object is copied on each node receiving a request for it. As explained in Section II, Figure 2(b), the object is written both on the first node it was written on and on the node that read it. In our approach using a Scale-out NAS, an object is written once for all. This prevents multiple duplications of one object on the same site. The object is only copied on another site in case of a remote access as described in Section III, Figure 4(c).

In other words, the default policy of IPFS consists in copying objects locally each time a node wants to access it can overpass largely the small side effect introduced by the fault tolerance capability of the underlying Scale-out NAS backend.

The limit of storage space reduction does not have any impact on the availability. The NAS replication/erasure coding allows individual node failures.

B. Performance Limitation and Future Work

From a performance viewpoint, it is important to underline that our current implementation can be improved. Indeed, when an IPFS node wants to access an object, stored remotely it retrieves the object from all the nodes having it (see Figure 2(b) in Section II). This feature comes from the BitTorrent protocol is interesting as it enables IPFS to mitigate the negative impact of overloaded nodes: If one node is overloaded, any other node storing the object will be able to satisfy the request in an efficient way.

In our approach using a Scale-out NAS, the situation differs because the DHT knows at most one node per site: either the node where the client has written the object or the first node used to retrieve the object in case of remote access. As described in Section III, Figure 4(c), the Site 3: IPFS Node4 can download the object only by contacting Site 1: IPFS Node1 because it is the original node that created the object (the DHT stores Site1:IPFS Node1 as the node owning the object). Site 3: IPFS Node4 does not know that Site1:IPFS Node2 has also access to the object stored in the Scale-out NAS. This situation can lead to bottleneck by overloading Site1:IPFS Node1.

Similarly, if Site1:IPFS Node1 crashes, the object will be unreachable from any node belonging to remote sites because IPFS Node1 is the only entry point for that object in the DHT. This is rather frustrating because the object is still available by contacting any other node belonging to Site1 (Site 1: IPFS Node2, for instance).

A solution for this problem may be not to add the address of a node in the DHT but an identifier of the site the object is. In this situation, remote nodes will be able to download the objects from all the nodes located on the sites the object is. Another solution can be placing all the IPFS nodes of a site at the same place in the DHT, making all the nodes of a site responsible for the same range of keys so that there is no distinction between the nodes of a site (as an example, Site 1: IPFS node1 and Site 1: IPFS node2 will have the same identifier). Evaluating pros/cons of each possibility is let as future work.

Despites these problems, the Scale-out NAS improves the availability of objects in case of network partitioning. If a site cannot reach the other ones, and therefore access to most of the nodes of the DHT, objects stored locally can still be retrieved by the clients connected to the site.

VI. Related works

There exists many approaches to manage the data location in an object store. But most of them either does not provide flexibility in data placement or they generate an important amount of network traffic by announcing the location of the objects to all nodes.

Using a hashing function or a placement function like CRUSH [22] is not easy because the topology has to be distributed in a consistent way between all the nodes. Also, the function does not provide a flexible way to specify the location of every object. Indeed, with CRUSH, to place explicitly each object, users have to specify placement rule for each of them which is not scalable.

Gossip protocols [23] enable this flexibility by distributing the location of each object to all nodes. But the network traffic depends on the number of stored objects. Also, nodes have to store the location of every object that may require a lot of storage space.

The approaches used in Cassandra [7] and Dynamo [24] combine the gossip and hashing. As an example, in Cassandra, the ranges of keys the nodes are responsible for is gossiped between the nodes and then nodes determine the locations of objects by hashing the object key. This approach is a trade-off between gossiping the location of each object and not be able to move the objects because of a hashing function.

The drawback of this approach is to manage the location of the objects by groups only. Placing each object independently leads to generate as much traffic as a traditional gossip.

VII. Conclusion

In this paper, we proposed to revise the IPFS proposal in order to cope with the Fog/Edge Computing context. Our proposal leverages a Scale-out NAS system deployed on each site in order to avoid the use of the global DHT when the requested object is available locally on the site. We implemented a Proof-of-Concept using RozoFS, an efficient Scale-out network file system. Evaluation using Grid'5000 showed that leveraging a Scale-out NAS on each site reduces the access times as well as the amount of network traffic sent between the sites. We also showed our approach does not impact the IPFS mobility criteria. Accessing an object from a remote site is also efficient in addition to reducing inter-site network traffic. The Scale-out NAS also provides IPFS an access to local files in case of network partitioning (when a site cannot reach the others) because the DHT is only accessed when a read is performed from a remote site. We discussed two additional aspects related to the storage space and possible performance improvements.

As a future work, we plan to investigate the benefit of coupling IPFS with a Scale-out NAS system where all the IPFS nodes of a site are considered as a single node in the DHT, avoiding performance and reliability problems previously described. Additionally, an evaluation in a scenario involving transient network and node failures should be done. Moreover, we would investigate the interest of placement strategies across the sites. One can envision to apply replication strategies between nearby sites. These strategies can be defined by end-users according to their requirements and the underlying network topology. Finally, gathering real I/O access patterns of Fog/Edge applictions is another goal. Indeed, such traces that are mandatory to evaluate Fog/Edge object store services are not yet available.

References

[1] F. Bonomi, R. Milito, J. Zhu, and S. Addepalli, "Fog Computing and Its Role in the Internet of Things," in *Proceedings of the First Edition of the MCC Workshop on Mobile Cloud Computing*, ser. MCC '12, 2012, pp. 13–16.

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

[3] A. Markopoulou, F. Tobagi, and M. Karam, "Loss and delay measurements of Internet backbones," *Computer Communications*, vol. 29, no. 10, pp. 1590 – 1604, 2006, Monitoring and Measurements of IP Networks.

[4] B. Zhang, N. Mor, J. Kolb, D. S. Chan, N. Goyal, K. Lutz, E. Allman, J. Wawrzynek, E. Lee, and J. Kubiatowicz, "The Cloud is Not Enough: Saving IoT from the Cloud," in *Proceedings of the 7th USENIX Conference on Hot Topics in Cloud Computing*, ser. HotCloud'15. Berkeley, CA, USA: USENIX Association, 2015, pp. 21–21.

[5] B. Confais, A. Lebre, and B. Parrein, "Performance Analysis of Object Store Systems in a Fog/Edge Computing Infrastructures," in *IEEE CloudCom*, Luxembourg, Luxembourg, Dec. 2016.

[6] S. A. Weil, A. W. Leung, S. A. Brandt, and C. Maltzahn, "RADOS: A Scalable, Reliable Storage Service for Petabyte-scale Storage Clusters," in *Proceedings of the 2nd International Workshop on Petascale Data Storage: Held in Conjunction with Supercomputing '07*, ser. PDSW '07, 2007, pp. 35–44.

[7] A. Lakshman and P. Malik, "Cassandra: A Decentralized Structured Storage System," *SIGOPS Operating Systems Review*, vol. 44, no. 2, pp. 35–40, Apr. 2010.

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

[9] A. Legout, G. Urvoy-Keller, and P. Michiardi, "Understanding BitTorrent: An Experimental Perspective," INRIA, Institut Eurecom, Tech. Rep., 2005.

[10] P. Maymounkov and D. Mazières, "Kademlia: A Peer-to-Peer Information System Based on the XOR Metric," in *Revised Papers from the First International Workshop on Peer-to-Peer Systems*, ser. IPTPS '01. London, UK: Springer-Verlag, 2002, pp. 53–65.

[11] D. Pertin, S. David, P. Évenou, B. Parrein, and N. Normand, "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.

[12] D. Balouek, A. Carpen Amarie, G. Charrier, F. Desprez, E. Jeannot, E. Jeanvoine, A. Lebre, D. Margery, N. Niclausse, L. Nussbaum, O. Richard, C. Pérez, F. Quesnel, C. Rohr, and L. Sarzyniec, "Adding Virtualization Capabilities to the Grid'5000 Testbed," in *Cloud Computing and Services Science*, ser. Communications in Computer and Information Science, I. Ivanov, M. Sinderen, F. Leymann, and T. Shan, Eds. Springer International Publishing, 2013, vol. 367, pp. 3–20.

[13] L. Mascetti, E. Cano, B. Chan, X. Espinal, A. Fiorot, H. G. Labrador, J. Iven, M. Lamanna, G. L. Presti, J. Mocicki, A. Peters, S. Ponce, H. Rousseau, and D. van der Ster, "Disk storage at CERN," *Journal of Physics: Conference Series*, vol. 664, no. 4, p. 042035, 2015.

[14] A. Lebre and G. Bervian Brand, "GBFS: Efficient Data-Sharing on Hybrid Platforms. Towards adding WAN-Wide elasticity to DFSes." in *WPBA Workshop in Proceedings of 26th International Symposium on Computer Architecture and High Performance Computing*, ser. WPBA Workshop in Proceedings of 26th International Symposium on Computer Architecture and High Performance Computing. Paris, France: IEEE, Oct. 2014.

[15] A. Davies and A. Orsaria, "Scale out with GlusterFS," *Linux J.*, vol. 2013, no. 235, Nov. 2013.

[16] S. Donovan, G. Huizenga, A. Hutton, C. Ross, M. Petersen, and P. Schwan, "Lustre: Building a file system for 1000-node clusters," in *Proceedings of the Linux Symposium*, 2003.

[17] R. D. Souza Couto, S. Secci, M. E. Mitre Campista, and L. H. Maciel Kosmalski Costa, "Network design requirements for disaster resilience in IaaS Clouds," *IEEE Communications Magazine*, vol. 52, no. 10, pp. 52–58, October 2014.

[18] J. Padhye, V. Firoiu, D. Towsley, and J. Kurose, "Modeling TCP Throughput: A Simple Model and Its Empirical Validation," *SIGCOMM Comput. Commun. Rev.*, vol. 28, no. 4, pp. 303–314, Oct. 1998.

[19] A. Miranda and T. Cortes, "Analyzing long-term access locality to find ways to improve distributed storage systems," in *Proceedings of the 2012 20th Euromicro International Conference on Parallel, Distributed and Network-based Processing*, ser. PDP '12. Washington, DC, USA: IEEE Computer Society, 2012, pp. 544–553.

[20] B. F. Cooper, A. Silberstein, E. Tam, R. Ramakrishnan, and R. Sears, "Benchmarking Cloud Serving Systems with YCSB," in *Proceedings of the 1st ACM Symposium on Cloud Computing*, ser. SoCC '10. New York, NY, USA: ACM, 2010, pp. 143–154.

[21] M. Imbert, L. Pouilloux, J. Rouzaud-Cornabas, A. Lebre, and T. Hirofuchi, "Using the EXECO toolbox to perform automatic and reproducible cloud experiments," in *1st International Workshop on UsiNg and building ClOud Testbeds (UNICO, collocated with IEEE CloudCom 2013*, Bristol, United Kingdom, Dec. 2013.

[22] S. A. Weil, S. A. Brandt, E. L. Miller, and C. Maltzahn, "CRUSH: Controlled, Scalable, Decentralized Placement of Replicated Data," in *Proceedings of the 2006 ACM/IEEE Conference on Supercomputing*, ser. SC '06, 2006.

[23] A.-M. Kermarrec and M. van Steen, "Gossiping in distributed systems," *SIGOPS Oper. Syst. Rev.*, vol. 41, no. 5, pp. 2–7, Oct. 2007.

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