2016 IEEE 8th International Conference on Cloud Computing Technology and Science

Performance Analysis of Object Store Systems in a Fog/Edge Computing Infrastructures

Bastien Confais
CNRS, IRCCyN UMR 6597,
1, rue de la Noë, BP 92101,
44321 Nantes Cedex 3, France

Adrien Lebre
INRIA, LINA UMR 6241,
Mines de Nantes,
4 Rue Alfred Kastler,
44300 Nantes, France

Benoît Parrein
Université de Nantes, IRCCyN UMR 6597,
Polytech Nantes,
rue Christian Pauc, BP 50609,
44306 Nantes Cedex 3, France

***Abstract*—Fog/Edge computing infrastructures have been proposed as an alternative of current Cloud Computing facilities to address the latency issue that prevents the development of several 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 couple of works illustrated the advantages of such infrastructures in particular for the 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 focused (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 sites during such operations. Experiments have been conducted using the Yahoo Cloud System Benchmark (YCSB) on top of the Grid'5000 testbed. We show that among the three tested solutions IPFS fills most of the criteria expected for a Fog/Edge computing infrastructure.**

I. Introduction

The advent of smartphones, tablets as well as Internet of Things (IoT) devices revolutionized the ways people are consuming IT services. While lots of applications take the advantage of the Internet and Cloud Computing solutions to extend devices' capabilities in terms of computations as well as storage, 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 [1]. The Fog Computing paradigm [2] 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.

Also known as Edge Computing, the advantages of such infrastructures have been described through several scenarios [3], [4]. However all these studies focused on particular use-cases and do not investigate whether generic Fog/Edge services can be envisioned. In this paper, we propose to address such a question for a online storage service. 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, a widely used service and a building block for hundred of Cloud services. We believe that such a storage service for Fog/Edge Computing infrastructures can pave the way toward new services as well as IoT applications.

9781-5090-1445-3/16$31.00 © 2016 IEEE (CloudCom'16)

The three storage systems we studied are Rados [5], the object storage module of the Ceph project, Cassandra [6], a high performance key value store and InterPlanetary File System (IPFS) [7], a storage system which uses the concepts brought by BitTorrent. We selected these three systems because they propose software abstractions for the definition of geographical *areas* without leveraging on a central server. We claim that such software-defined areas are important for the Fog/Edge Computing because it enables the storage system to consider the topology of the infrastructure (*i.e.*, one software area corresponds to one particular micro DC, that is one Fog/Edge site). This topology enables the development of advanced strategies to favor local accesses with the goal of mitigating the traffic that is exchanged between each site as well as the impact each site may have on the others.

The main contribution of our work is a deep performance evaluation enabling the identification of pros/cons of the strategies implemented by those three systems. Experiments have been done on top of Grid'5000 [8] by considering several data manipulation scenarios leveraging Yahoo Cloud Service Benchmark (YCSB) [9], a well-known benchmark tool designed to benchmark object stores [10], [11]. Our goal is to determine in an empirical way the lowest access time we can get for each storage system. To this end, we disabled mechanisms such as data replication and we modified some implementations like removing the metadata replication in IPFS.

The remaining of the paper is organized as follows. Section II defines the Fog/Edge computing model we consider and gives a list of characteristics a S3-like service should have in such a context. Section III presents an overview of the three storage systems. Evaluations are discussed in Section IV. Finally, Section V concludes this study and highlights some perspectives.

2330-2186/16 $31.00 © 2016 IEEE
DOI 10.1109/CloudCom.2016.52

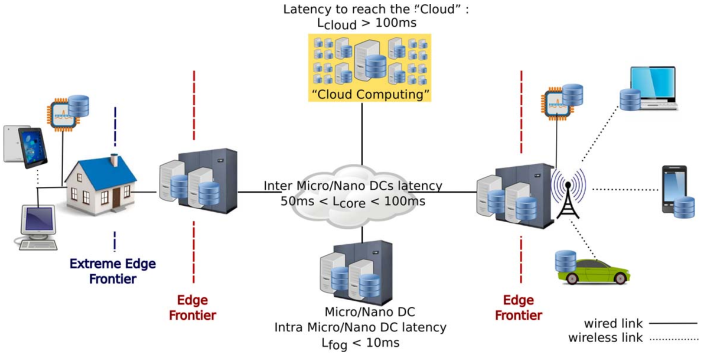


Fig. 1. Overview of a Cloud, Fog and Edge platform.

II. Fog/Edge Computing model

In this section, we present the Fog/Edge architecture we are considering and a list of characteristics we claim an object storage system should have in such a context.

A. *Fog/Edge Network Topology*

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 [1], [12]. Moreover, Fog sites can complement each other to satisfy the needs between user's devices and Cloud Computing centers [3], [13]. 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. Each site hosts a limited number of servers that offer storage and computing capabilities. End-users devices (smartphones, tablets, laptops) as well as IoT devices can reach a Fog site with a rather low latency (noted L_{fog}) lesser than 10 ms (latency of a wireless link). We consider that the latency between Fog sites (noted L_{core}) is up to 50 ms (mean latency of a Wide Area Network link [14]). The latency to reach a Cloud platform (noted L_{cloud}) from the clients is higher (about 200 ms) [15], [16] and unpredictable [1].

B. *Storage requirements*

Our objective is to study how a storage service such as a S3 object store system should be designed to deal with and take the advantage of Fog/Edge Computing infrastructure specifics. In addition to the scalability requirement that is intrinsic to such a massively distributed infrastructure, we advocate that a Fog/Edge computing storage service should meet the following properties:

- low access time (by favoring data locality);
- network containment between sites;
- availability of data in case of service partitioning;
- support for users mobility.

Low access time is the main characteristic behind the motivation for the Fog paradigm. The idea is to favor local accesses each time it is possible (*i.e.*, each `put` into the object store should be handled by the closest site, assuming that each Fog site delivers the same storage performance).

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 criteria can be delivered by mitigating data transfers between sites each time an operation is performed.

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 or coding techniques such as XORing Elephants [17] designed for multi-sites environments can ensure data availability.

we claim that a Fog storage service should be able to run, at least, in a degraded mode in order to tolerate local accesses and provide appropriate mechanisms to reconsolidate the service once the disconnection is completed.

Mobility support is the last property we have identified. The idea is to enable data to transparently follow 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.

Last but not the least, we underline that discussing consistency model of the storage service we target is behind-the-scope of this first study. Related to the objective of having the lowest latency as possible, we consider objects like files and documents of one user with no parallel accesses.

III. "Off-the-shelf" Distributed storage systems

Distributed file systems such as PVFS [18], Lustre [19], HDFS [20] and other WANWide-like proposals [21], [22] are not appropriated to the Fog/Edge Computing infrastructure we target because they all have been designed around the concept of an entity in charge of maintaining the storage namespace in a rather "centralized" manner. Because they are leveraging P2P mechanisms, object and key/value store systems are better candidates to cope with the requirements we defined in the previous section. Among the different solutions that are available, we selected Rados [5], Cassandra [6], two storage systems developed for the Cloud and IPFS [7] designed to work at the edge of the network. We chose these systems because they provide software abstractions that enable the definition of areas that can be mapped to geographical sites. We present in the following paragraphs, an overview of these three systems. Storage systems like Dynamo [23] have not been studied because their conception is similar to Cassandra, but the location of every object is gossiped between the nodes.

A. *Rados*

Rados [5] is an object distributed storage solution which uses the CRUSH algorithm [24] to locate data in the cluster without requiring any remote communication from the clients.

Rados uses two kinds of nodes: Object Storage Daemons (OSD) and Monitors. OSDs store objects whereas the monitors maintain a tree (*a.k.a.* the "clustermap") describing the cluster's topology. Among the monitors, one is elected as the master and is in charge of maintaining a consistency view of the clustermap (the Paxos algorithm is used to guarantee that there is only one master monitor). The "clustermap", that is distributed to each client node is used by the CRUSH algorithm to locate objects.

Rados allows administrators to organize objects within pools. Each pool is associated to "placement rules". In our context, we use placement rules to constraint objects of a particular pool to be located on one site. To write or read an object, clients must provide the name of the pool.

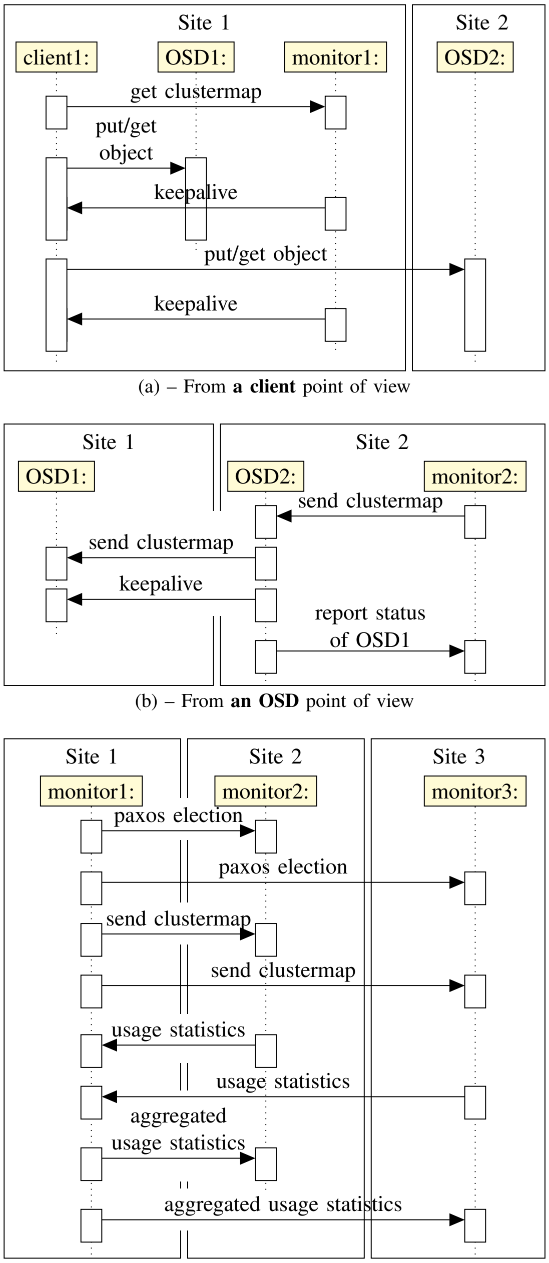
When the network is partitioned, only the data located in the partition where a master monitor can still be elected, is available. In other words, clients that belong to this partition can access the "clustermap" and thus any object that is reachable. On the other partitions, because clients cannot get the "clustermap" they cannot locate and thus access any object.

Figure 2 shows the sequence diagram of exchanges we observe in a Rados deployment, respectively from a client (a), an OSD (b) and a monitor (c) point of view. In addition to the exchanges we described, Rados uses a large number of keepalives messages between the different nodes. This enables the system to swiftly react in case of failures.

In overall, the inter-site overhead is a mix of Paxos' messages, distributions of clustermap, and usage statistics sent between the monitors. The inter-site covers also some report of OSD status. To mitigate as much as possible this overhead, it is possible to place one monitor per site. This minimizes the report of OSD status between sites as well as the overhead related to the clustermap retrievals.

B. *Cassandra*

Cassandra is a key value store designed on a one-hop distributed hash table (DHT) [6]. Cassandra is organized in a one-hop DHT. The set of values the objects' key are defined 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. Once gossiped data are received, storage nodes can locate any object without any extra communication, just by hashing the object's key and contacting the node responsible for.



(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

Fig. 2. Sequence diagrams of the network traffics observed in Rados.

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

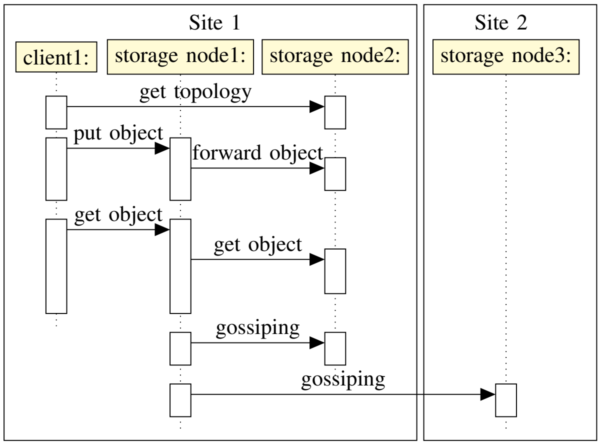


Fig. 3. Sequence diagrams of the network traffics observed in Cassandra.

consistency. Moreover, Cassandra exposes the same notion of pools as the one proposed by Rados. Entitled “keyspaces”, each pool is associated to a placement strategy. The “NetworkTopologyStrategy” for instance enables administrators to specify for each data-center (and in our case, for each site) how many replicas of an object they want to store. For the purpose of our analysis, we defined a strategy where there is 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 each time an object is pushed into it.

Like Rados, users have to specify the keyspace’s name they want to use. The major message exchanges related to the Cassandra protocol are illustrated on Figure 3: Clients retrieve the topology from the server they connect to. Based on the topology they retrieved, they contact one of the nodes of the site they belong to. For performance purposes, requests are balanced in a round-robin way. That is, a client sends requests alternatively to the different servers located on the site. The request is then forwarded to the server, which has to handle it (the server is determined based on the aforementioned strategy). Thanks to the “NetworkTopologyStrategy”, the gossip traffic is the only overhead going throughout the different sites.

C. InterPlanetary File System

InterPlanetary File System [7] has a simpler architecture than Rados and Cassandra. It has been built on the BitTorrent protocol [25] and the Kademlia DHT [26], both being well-known protocols for their ability to scale to a large number of nodes. These technologies come from the P2P world. 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 comparisons to Rados and Cassandra that have designed dedicated mechanisms to locate objects without extra communication from the clients viewpoint.

Figure 4 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 that 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. Finally, the location of the new replica is announced in the DHT. Thanks to such an approach, IPFS supports the mobility of data in a native fashion. A future read will be satisfied by this node, closer to the client. Storage nodes send regularly keepalive messages to maintain the DHT.

It is noteworthy that IPFS uses immutable objects. Modifying an existing object creates a new one. Because objects are immutable, it is easier to maintain the consistency between all replicas of one object. Moreover, the BitTorrent protocol that is used to pull the data enables to retrieve the same object from several sources simultaneously.

Last but not the least, 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.

D. Summary

Table I summarizes how do Rados, Cassandra and IPFS fit the requirements that have been defined in Section II-B. The data locality is satisfied by Rados and Cassandra by “placement rules” and by “placement strategies”.

The network containment is satisfied by Rados, thanks to the CRUSH algorithm. The Cassandra gossiping distributes only the ranges the nodes are responsible for, thus, inserting an object has no impact on the gossip traffic. But for IPFS, each objects is announced in the DHT.

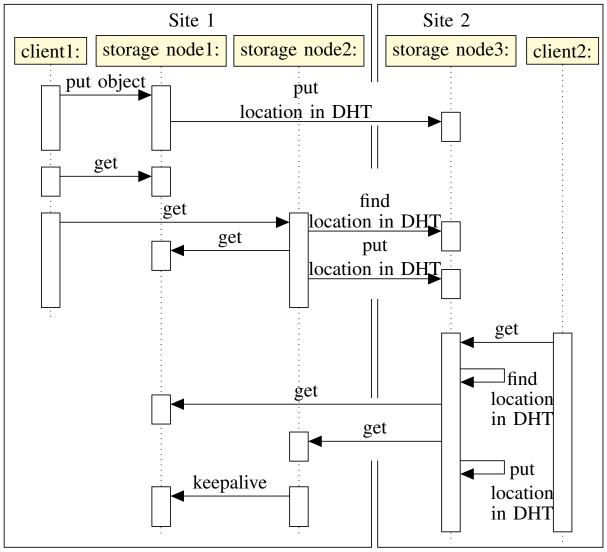


Fig. 4. Sequence diagram of the network traffics observed in IPFS from **a client** point of view. The read from “client2” can be performed using “storage node2” only after “client1” read from “storage node2”.

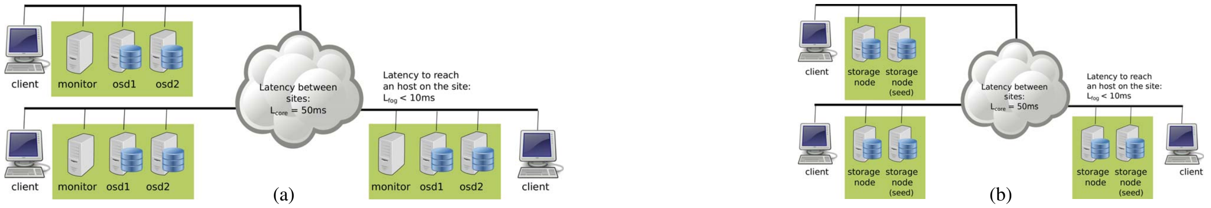

client
monitor
osd1
osd2
Latency between sites: L_{core} = 50ms
Latency to reach an host on the site: L_{fog} < 10ms
client
monitor
osd1
osd2
monitor
osd1
osd2
client
(a)
client
storage node
storage node (seed)
Latency between sites: L_{core} = 50ms
Latency to reach an host on the site: L_{fog} < 10ms
client
storage node
storage node (seed)
storage node
storage node (seed)
client
(b)

Fig. 5. Topology used to deploy Rados (a), Cassandra and IPFS (b) in a Fog environment.

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

TABLE I
SUMMARY OF FOG CHARACTERISTICS *a priori* MET FOR 3 DIFFERENT OBJECT STORES.

The lack of locality in the DHT for IPFS and the Paxos algorithm for Rados limits the data availability in case of service partitioning (and the scalability for Rados). Finally, the partial support of the mobility property comes from the explicit relocation needed. Based on the specifications, IPFS seems “a-priori” the best candidate for the Fog, although problems such as support of network partitioning needs to be solved.

IV. EXPERIMENTS

In this section, after describing the settings used for experimentation in Section IV-A, we discuss evaluations we performed on the three systems. The first experiment (Section IV-B) focuses on local access times and on the amount of network traffic exchanged between sites. The second one (Section IV-C) measures the remote access time, when data are accessed from a site which does not store it.

A. Material and method

The Fog architecture as well as the way we deployed the different systems are presented in Figure 5(a). For Rados, each site hosts one client, two storage servers (small sites) and one monitor. With Cassandra and IPFS, one server per site is configured as a seed node, so that other nodes can join the cluster without inter-site communications. With Rados each pool is associated to a placement rule to force data to be stored on a specific site and with Cassandra, the ”NetworkTopologyStrategy” is used as previously described in Section III-B. With IPFS, locality is natively supported because data are located on the node the client writes on.

Experiments are performed on the Grid’5000 testbed using the “Paravance” cluster.Latencies between hosts are emulated using the `tc` program. We use $L_{fog} = 10\,\text{ms}$ and $L_{core} = 50\,\text{ms}$ (as depicted in Figure 1). The throughput of the network links is 10 Gbps. We have simplified the network model by considering only the latency in this work. We did not considered the bandwidth as a bottleneck. According to the Nielsen’s law [27], the bandwidth available matches the bandwidth usage.

The metrics we measure are the time taken to perform operations on each site and the amount of network traffic sent between the sites on that period. Flushing operation (`sync`) is not taken into account in the measurements.

Regarding IPFS and still in order to ensure fair comparisons between the three systems, requests have been balanced between the servers of a same site. This means that one object could have been written on one server and then a read request can be sent to the other server of the site (resulting with an additional read copy on the server as described in Section III-C) Moreover, IPFS adds redundancy in the Kademlia DHT: for each object, the location is stored on 10 different nodes to support failures. We modify IPFS to avoid this replication of metadata.

The Yahoo Cloud System Benchmark [9] (YCSB) has been used to emulate data accesses. We use different access patterns in order to check if the storage systems may be used for different kinds of applications. Object sizes vary from 1 MB to 10 MB. Scenarios have been performed using 1, 7 and 11 sites. Each experiment has been performed at least 10 times (10 trials) to get consistency in results and stored data are erased between the trials. Grid’5000 provides a stable environment to perform the experiments: the network latency does not vary during the experimentation.

Cassandra and Rados modules for YCSB already exist but we wrote the module for IPFS. The measured access times for Cassandra are high compared to what we measure with our “home-made” benchmark script. Indeed, the java garbage collector is called when the connection to the cluster is closed, increasing a lot the access times. Therefore, we decided to remove this instruction. Writes are performed during the load phase of YCSB and reads during the run phase. With the used workload, clients connect the storage systems, execute several read or insert operations and then close their connection. Neither update nor delete operations are performed. Read objects are selected randomly, according to a uniform distribution.

B. Local read/write evaluation (experiment 1)

All clients (one per site) execute the access scenario simultaneously: they write objects on their site and read them. The goal is to evaluate data locality as well as network containment properties on each site.

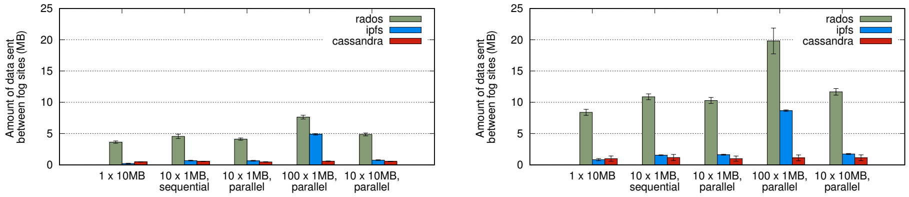
1) Writing and reading times: Table II shows the access times measured with YCSB. All standard deviations are less than 0.6 seconds and thus have not been reported.

The first observation is with Rados, access times do not increase significantly when sites are added. As an example, it takes 3.15 seconds to write one object of 10 MB on 1 site

columns		1	2	3	4	5	6	7	8	9	10
rows	Objects	**Mean writing time** on a site (seconds)					**Mean reading time** on a site (seconds)				
		1× 10 MB	10× 1 MB sequential	10× 1 MB parallel	100× 1 MB parallel	10× 10 MB parallel	1× 10 MB	10× 1 MB sequential	10× 1 MB parallel	100× 1 MB parallel	10× 10 MB parallel
		1 sites									
1	Rados	3.15	4.48	2.28	2.42	3.51	3.17	4.26	2.17	2.40	2.97
2	Cassandra	3.87	9.56	3.56	5.60	7.42	3.91	9.09	3.54	5.36	5.76
3	IPFS	2.30	2.75	1.95	2.77	2.80	1.58	1.94	1.24	2.02	2.06
		7 sites									
4	Rados	3.27	5.02	2.28	2.46	3.83	3.23	4.92	2.21	2.41	3.84
5	Cassandra	4.97	9.73	4.24	6.65	7.93	4.91	9.64	4.22	6.08	7.05
6	IPFS	2.33	2.55	1.93	2.74	2.87	1.45	1.94	1.11	2.02	2.04
		11 sites									
7	Rados	3.67	5.70	2.38	2.78	3.68	3.31	5.01	2.26	2.43	3.66
8	Cassandra	6.25	10.45	4.94	7.71	8.25	6.12	10.64	4.90	6.69	7.21
9	IPFS	2.44	2.61	1.92	2.80	2.67	1.47	1.92	1.34	2.02	2.11

TABLE II

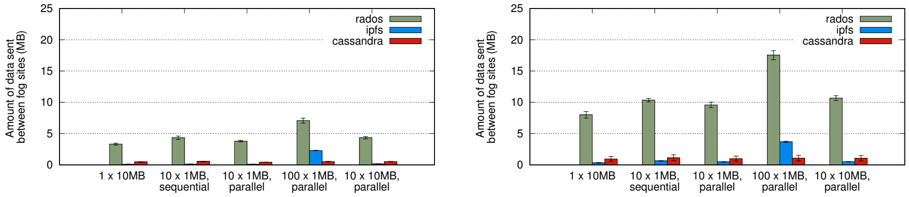
MEAN TIMES **IN SECONDS** TO WRITE AND READ ALL THE OBJECTS ON **A SITE** WITH 1, 7 AND 11 SITES.



(a) – 7 sites

(b) – 11 sites

Fig. 6. Cumulated amount of network traffic exchanged between all the sites while clients **write** objects on their sites.



(a) – 7 sites

(b) – 11 sites

Fig. 7. Cumulated amount of network traffic exchanged between all the sites while clients **read** objects located on their sites.

and 3.67 seconds on 11 sites. This comes from the use of the CRUSH algorithm: read or write on a site does not require any communication with nodes outside the site (the clustermap is obtained from the local monitor). Also, access times are similar for writing and reading. For example, it takes 2.78 seconds to write 100 objects on 11 sites and 2.43 seconds to read them. Indeed, the process to read or write is the same: execute the placement function on the object's key and access the OSD storing it.

With Cassandra, it takes 0.94 seconds to write 100 objects of 1 MB one on site without extra latencies (not presented in the Table II). However, with $L_{fog} = 10\,\text{ms}$, it takes 5.60 seconds (row 2, column 4). The forward mechanism due to the balance of requests between nodes (see Section III-B) also increases the access times. More generally, access times are increasing linearly with the number of sites. The gossip traffic between the nodes (described in the next section) has an influence on the access times. Adding more sites generates more gossip traffic and longer access times. Because there is no churn in the network and thus no need to gossip a topology change, we expected Cassandra gets better access times. Also, we measured high connection times for the clients to connect the Cassandra cluster (approximately 2 seconds).

With IPFS, access times do not increase a lot when there are additional sites. For instance, it takes 2.30 seconds to write one object of 10 MB on 1 site (row 3, column 1) and 2.44 with 11 sites (row 9, column 1). Reading 100 objects of 1 MB, takes strictly the same time for 1, 7 and 11 sites (2.02 seconds) (rows 3, 6 and 9, column 9). Indeed, because there are more nodes, even if there are more DHT requests, each node does not have more DHT requests to process.

2) Amount of network traffic sent between sites: Figures 6 and 7 show the amount of network traffic sent between sites. Error bars correspond to the standard deviation. The first comment is that for each system, the traffic that is exchanged between sites is relatively small in comparison to the amount of data stored (less than 2% of the amount of stored data). For Rados, this traffic is the most important (7.5 MB both in writing and in reading for 100 × 1 MB on 7 sites) and increases with the number of monitors (*i.e.* the number of sites) with the same order of magnitude for writing and reading. As previously described, all monitors send usage statistics to the elected monitor which merges and redistributes them. This strategy becomes very costly when there is a huge number

columns		1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
rows		**Mean writing time** on a local site (seconds)					**Mean reading time** on a remote site (seconds)									
							First read					Second read				
	Objects	1× 10 MB	10× 1 MB sequential	10× 1 MB parallel	100× 1 MB parallel	10× 10 MB parallel	1× 10 MB	10× 1 MB sequential	10× 1 MB parallel	100× 1 MB parallel	10× 10 MB parallel	1× 10 MB	10× 1 MB sequential	10× 1 MB parallel	100× 1 MB parallel	10× 10 MB parallel
1	Rados	3.19	4.49	2.24	2.53	3.53	15.36	26.12	6.06	5.11	10.35	14.58	27.48	6.97	5.18	10.19
2	Cassandra	3.37	10.01	3.49	6.27	6.76	13.90	30.92	9.12	20.97	26.88	14.06	30.75	9.11	18.66	26.28
3	IPFS	2.77	3.50	1.92	2.75	2.37	4.40	9.27	2.10	6.27	12.48	1.16	2.60	2.02	2.33	2.33

TABLE III

Mean times **in seconds** to write all objects on a site and read them twice from a remote site. There are 7 sites but only one site is used to write and another one to read.

of monitors. The amount of metadata sent between the sites increases linearly with the number of objects, regardless their size. Indeed, this is because usage statistic packets have a size proportional to the number of accessed objects. This explains also the fact that we observe more data exchanged between the sites when 10 objects are written in a sequential way than in a parallel way. When objects are sent in a sequential way, more statistics are sent because the test is longer. The difference of exchanged data comes from the headers of statistics packets which are more important in this case. These metadata are sent asynchronously because these traffics do not impact the access times. As discussed in the Section III-A it is possible to reduce the number of monitors (in our testbed we used one monitor per site). While such a strategy will decrease the network overheads related to monitors, it will increase the traffic related to the OSD status reporting and clustermap retrievals. We know Paxos is not a scalable algorithm and these exchanges of usage statistics confirm us that the number of monitors should be kept low.

With Cassandra, the amount of traffic is increasing in a linear way with the number of nodes because each second nodes sends a gossip packet to another one, as explained in Section III-B.

Finally, the amount of metadata sent with IPFS is linear to the number of manipulated objects, regardless their sizes. While reading, IPFS sends less traffic between sites than in writing. There is no lookup in the DHT for every requested object but only for requests addressed to a node which does not store the object. In this case, a put is also done to announce its new location. Also, an object can be requested several times because of the uniform request distribution but once the object is located on all the nodes of a site, the DHT is not be consulted anymore. If we add more storage nodes on the sites, clients will have a lower probability to request a node storing the data, and more often, the DHT will be used, generating more network traffic between sites.

3) Summary: In this experiment, Rados gets similar performance in reading and in writing, but the amount of network traffic sent between the monitors can become a scaling difficulty. Cassandra gets the highest access times, but it is not related to the location mechanism. It is due to the time the clients spend to connect the cluster. For IPFS, The main drawback is the global DHT, which generates more network traffic between the sites and higher access times when the number of accessed objects is increased.

C. Remote reading evaluation (experiment 2)

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

One client creates 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. Read objects are not exactly the same between the two read, due to the uniform request distribution used in YCSB. 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). We just evaluate the time to perform a read remotely.

Table III shows the access times we get in this scenario. Standard deviations are all less than 0.8 seconds and thus still not reported as in the first experiment. We observe the writing time is the same as in the previous experimentation when data are written on only one site (see row 1 to 3, columns 1 to 5 in Tables II and III). We compare the times with the version using one site because here, only one client writes.

Regarding read accesses, for the first ones (columns 6 to 10), the Rados client contacts directly the OSD storing the requested object. So the increasing of access time in reading is only due to the network latency L_{core} between the client and the storage node. It takes 15.36 seconds (row 1 column 6) to read 1 object of 10 MB remotely. This is roughly five times the time we measured in the previous experiment (3.17 seconds, row 1 column 6 in Table II)). 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 in Figures 3 and 4). In this experiment, because objects are stored remotely, the forward is performed for all objects while in the previous experiment, only the requests addressed to the node which does not store the object required a forward.

For Cassandra, a remote read takes 13.90 seconds (row 2, column 6) whereas it lasted only 3.91 seconds in the first experiment (row 2, column 6 in Table II). The metadata management does not imply an increasing of access time because, once the gossip propagated, every node can locate any object without any communication. For IPFS, a remote read for $1 \times 10\,\mathrm{MB}$ takes 4.40 seconds whereas a local read took 1.58 seconds (row 3, column 6 in Table II and III).

The last part of Table III (columns 11 to 15) shows in the second read, the access times for IPFS are in the same order of magnitude than in the first experiment. Namely, IPFS spends 2.33 seconds to read 100 object of 1 MB (vs 2.02 s in experiment 1). Access times are low because the requested nodes serve a copy of the objects they kept in the first read. This second read is a local read for most of the objects and a remote read for objects that were not requested in the first one. For Rados and Cassandra, access times do not evolve as expected. Indeed, the mobility of data are not explicit and thus the second access generates a remote read also. It might be possible to get similar behaviour by explicitly changing the placement rule of the pool (or the keyspace). Once again, IPFS delivers the best performance.

V. Conclusion and Future work

In this work, we presented a list of characteristics 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). The performance is measured in terms of latency and network traffic. Globally, Rados (because of the Paxos traffic) and Cassandra (because the access time is proportional to the nodes number) encounter difficulties to scale in the Fog context. Although is architecture is simpler, IPFS seems to provide lowest latencies both for local and remote accesses. The monitored inter-site traffic is moreover relatively contained. However, the Kademlia DHT used in IPFS to ensure metadata management is not well suited for local activity expected in a Fog context. A locality-aware management of the meta-data within IPFS will be proposed in a future work to address this problem.

VI. Acknowledgement

Authors would like to thank the reviewers for their relevant remarks that allowing us to improve the quality of the paper.

References

[1] 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, 2015.

[2] 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.

[3] H. Dubey, J. Yang, N. Constant, A. M. Amiri, Q. Yang, and K. Makodiya, "Fog Data: Enhancing Telehealth Big Data Through Fog Computing," in *Proceedings of the ASE BigData & SocialInformatics 2015*, ser. ASE BD&SI '15, 2015, pp. 14:1–14:6.

[4] M. Aazam and E.-N. Huh, "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*, ser. FICLOUD'14. Washington, DC, USA: IEEE Computer Society, 2014.

[5] 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*, 2007.

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

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

[8] D. Balouek, A. Carpen Amarie, G. Charrier, F. Desprez, E. Jeannot, E. Jeanvoine, A. Lèbre, 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.

[9] 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.

[10] V. Abramova and J. Bernardino, "NoSQL Databases: MongoDB vs Cassandra," in *Proceedings of the International C* Conference on Computer Science and Software Engineering*, ser. C3S2E '13. New York, NY, USA: ACM, 2013, pp. 14–22.

[11] V. Abramova, J. Bernardino, and P. Furtado, *Evaluating Cassandra Scalability with YCSB*. Cham: Springer International Publishing, 2014.

[12] F. Bonomi, R. Milito, P. Natarajan, and J. Zhu, "Fog computing: A platform for Internet of Things and analytics," in *Big Data and Internet of Things: A Roadmap for Smart Environments*, ser. Studies in Computational Intelligence, N. Bessis and C. Dobre, Eds. Springer International Publishing, 2014, vol. 546, pp. 169–186.

[13] C. C. Byers and P. Wetterwald, "Fog Computing Distributing Data and Intelligence for Resiliency and Scale Necessary for IoT: The Internet of Things," *Ubiquity*, vol. 2015, pp. 4:1–4:12, Nov. 2015.

[14] 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.

[15] 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.

[16] M. Firdhous, O. Ghazali, and S. Hassan, "Fog Computing: Will it be the Future of Cloud Computing?" in *Third International Conference on Informatics & Applications, Kuala Terengganu, Malaysia*, 2014.

[17] M. Sathiamoorthy, M. Asteris, D. Papailiopoulos, A. G. Dimakis, R. Vadali, S. Chen, and D. Borthakur, "XORing Elephants: Novel Erasure Codes for Big Data," *Proc. VLDB Endow.*, vol. 6, no. 5, pp. 325–336, Mar. 2013.

[18] P. H. Carns, W. B. Ligon L, R. B. Ross, and R. Thakur, "PVFS: A Parallel File System for Linux Clusters," in *Proceedings of the 4th Annual Linux Showcase & Conference - Volume 4*, ser. ALS'00. Berkeley, CA, USA: USENIX Association, 2000, p. 28.

[19] 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.

[20] K. Shvachko, H. Kuang, S. Radia, and R. Chansler, "The Hadoop Distributed File System," in *Proceedings of the 2010 IEEE 26th Symposium on Mass Storage Systems and Technologies (MSST)*, ser. MSST '10. Washington, DC, USA: IEEE Computer Society, 2010, pp. 1–10.

[21] F. Hupfeld, T. Cortes, B. Kolbeck, J. Stender, E. Focht, M. Hess, J. Malo, J. Marti, and E. Cesario, "The XtreemFS architecture a case for object-based file systems in Grids," *Concurrency and computation: Practice and experience*, vol. 20, no. 17, pp. 2049–2060, 2008.

[22] O. Tatebe, K. Hiraga, and N. Soda, "Gfarm grid file system," *New Generation Computing*, vol. 28, no. 3, pp. 257–275, 2010.

[23] 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.

[24] 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.

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

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

[27] J. Nielsen, "Nielsens law of Internet bandwidth," 1998.