Improving locality of an object store in a Fog Computing environment

Bastien Confais, Benoît Parrein, Adrien Lebre
LS2N, Nantes, France

Grid'5000-FIT school

4th April 2018

Outline

1. Fog computing architecture
2. Improving locality when accessing an object stored locally
3. Improving locality when accessing an object stored on a remote site
4. A more realistic experiment using FIT and G5K platforms
5. Conclusion

Fog Computing architecture

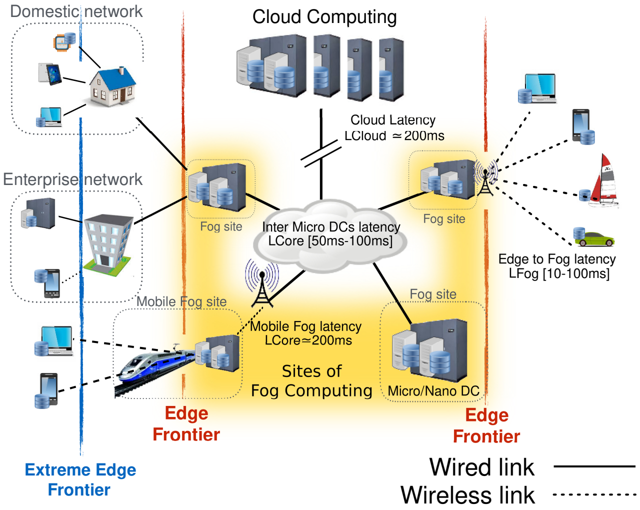


Figure 1: Overview of a Cloud, Fog and Edge infrastructure.

Properties for a Fog Storage system

We established a list of properties a distributed storage system should have:

- Data locality;
- Network containment;
- Mobility support;
- Disconnected mode;
- Scalability.

Assumptions

- Clients use the closest Fog site;
- $L_{Fog} \ _{(\approx 10\ ms)} \leq L_{Core} \ _{(\approx 100\ ms)} \leq L_{Cloud} \ _{(\approx 200\ ms)}$; enlever
- Objects are immutable;
- We want to access the closest object replica;
- We particularly focus on location management.

IPFS in a nutshell

Among three existing object stores, InterPlanetary File System (IPFS)[1] filled most of the properties (Rados and Cassandra were also studied)[2].

IPFS is an object store that uses:

- a Kademlia Distributed Hash Table (DHT) spread among all the nodes to locate the objects;
- a BitTorrent like protocol to exchange the objects.

[1] **DBLP:journals/corr/Benet14**

[2] **confais:hal-01397686**

Improving locality when accessing an object stored locally

Reading an object stored locally

Limitation When the requested node does not store the object, Inter-sites network traffic is generated by accessing the DHT to locate it (in red).

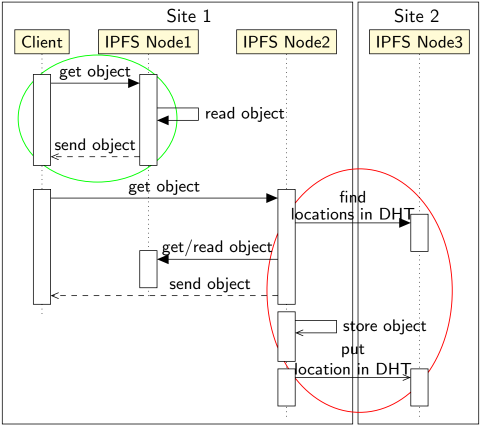


Figure 2: Network exchanges when a client **reads** an object stored **locally, on IPFS Node1**.

Our solution: coupling IPFS and a Scale-Out NAS[3]

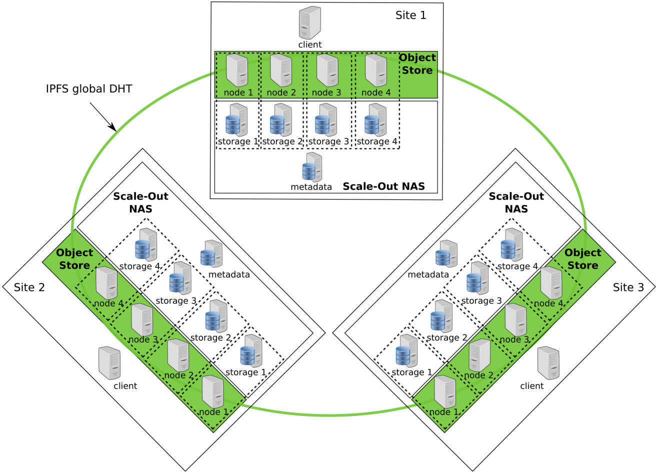


Figure 3: Topology used to deploy an object store on top of a Scale-Out NAS local to each site.

[3] **ICFEC2017**

Reading an object stored locally using IPFS and a Scale-Out NAS

New protocol behaviour

The global DHT is not accessed because all the nodes of the site can access all the objects stored on the site. The object is read from the Scale-Out NAS.

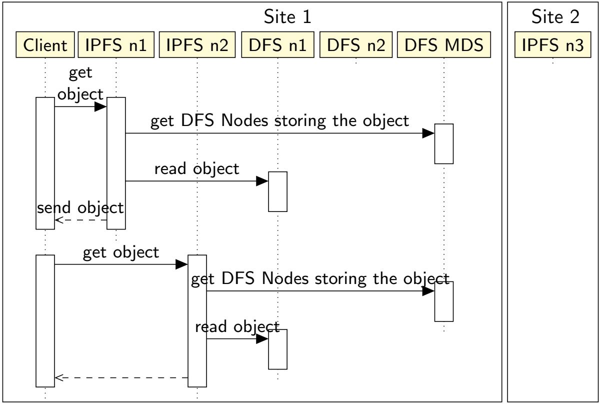


Figure 4: Network exchanges when a client **reads** an object stored **locally**.

Experimental Evaluation

We evaluate only on the **Grid'5000** testbed 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.

In two scenarios:

local access	one client on each site writes and reads objects stored locally;
remote access	one client on one site writes locally and another client located on another site reads it.

We use RozoFS[4] as Scale-Out NAS and a `tmpfs` as a low level backend.

[4] **pertin:hal-01149847**

Material and Methods

We measure:

Average access time: the average time to write or read an object in a specific workload;

Network traffic: the amount of data exchanged between the sites.

The (one-way) latencies between the different nodes have been set in order to be representative to:

- local wireless link, $L_{Fog} = 10\,\mathrm{ms}$;
- wide area network link, $L_{Core} = 50\,\mathrm{ms}$;
- the latency to reach the cloud, $L_{Cloud} = 100\,\mathrm{ms}$;
- the latency between the server of a same site: $0.5\,\mathrm{ms}$.

Our benchmark code as well as raw results are available at
`https://github.com/bconfais/benchmark`

Average access times while writing and reading from the same site

	Mean **writing** time (seconds)				Mean **reading** time (seconds)			
	Size Number	256 KB	1 MB	10 MB	Size Number	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)			
	Size Number	256 KB	1 MB	10 MB	Size Number	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)			
	Size Number	256 KB	1 MB	10 MB	Size Number	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 1: **Mean time** (seconds) to write or read one object under different conditions (the number on the left indicates the number of operations executed in parallel on each client).

Advantages & Drawbacks

Our approach has several advantages:

Contains the network traffic: The DHT is only used for remote accesses;

Increases locality: Local replicas are first accessed (before remote ones);

But also a drawback:

DHT does not reflect the actual location: A remote site can only access the object through the node it was written on, and not from all the nodes of the site;

Improving locality when accessing an object stored on a remote site

Reading an object stored remotely

Limitation A third site is potentially solicited due to the DHT repartition, and this first site is not necessarily close to the client.

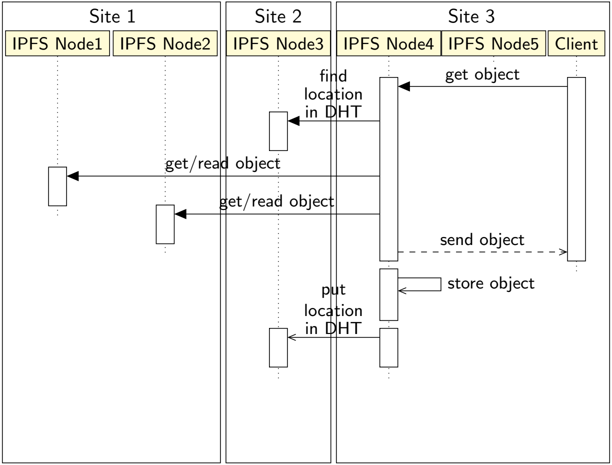


Figure 5: Network exchanges when a client **reads** an object stored on a **remote** site (read from Node4).

Drawbacks of the DHT

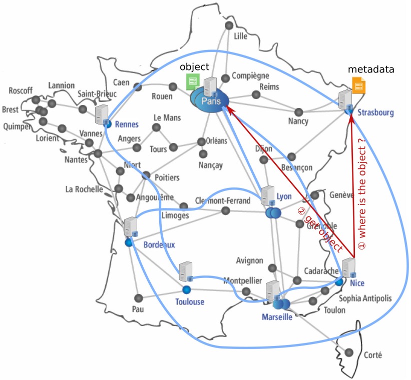


Figure 6: Exchanges when an object stored in Paris is accessed from Nice.

- The DHT overlay is built according to random node identifiers that do not map the physical topology. For instance, Rennes and Strasbourg are neighbours in the DHT but are not close physically (Paris is between them).
- Because of the consistent hashing used in the DHT, Nice needs to contacts Strasbourg to locate an object actually stored in Paris.

Drawbacks of the DHT

DHT does not take into account the physical topology: Neighbours in the DHT may be physically far and the latency between them may be high;

DHT prevents locality: Strasbourg has to be contacted although it is not concerned by the objects. Accessing location record may be done with a higher latency than the latency to access the object.

DHT prevents disconnected mode: If Strasbourg is unreachable, Nice cannot reach the object stored in Paris;

An inspiration from the DNS protocol

Our approach is inspired by the Domain Name System (DNS). In the DNS, a resolver sends requests from the root node to node which actually stores the information needed.

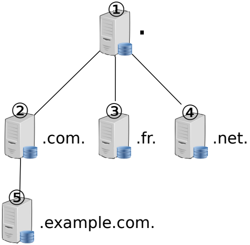
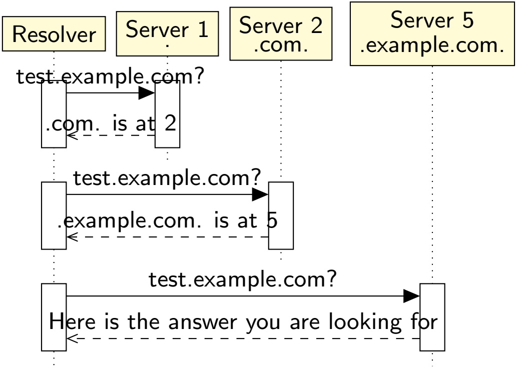


Figure 7: Example of a DNS Tree



Resolver

Server 1 .

Server 2 .com.

Server 5 .example.com.

test.example.com?

.com. is at 2

test.example.com?

.example.com. is at 5

test.example.com?

Here is the answer you are looking for

Figure 8: Messages exchanged during an iterative DNS resolution.

A Tree based Metadata Replication Strategy

We propose to store the object's location in a tree.

The tree is built according to the physical topology so that the parent of a node is reachable with a lower latency than the parent of its parent.

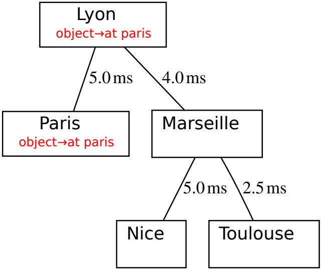


Figure 9: Example of subtree computed with our algorithm.

In Figure **??** Toulouse is closer to Marseille than Lyon.

Locations of objects are stored in all nodes at the top of the node storing a replica (the location of an object stored at Paris is also stored at Lyon).

Read protocol (1/3)

Contrary to the DNS, requests are sent **from the current node towards the root**:

- to first request the node reachable with the lowest latency;
- to locate the closest replica;
- to enable disconnected mode.

Read protocol (2/3)

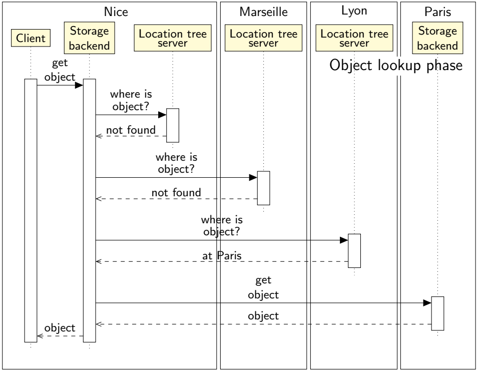


Figure 10: Read the object stored in Paris from Nice.

Metadata is found at Lyon, which is the root of the tree but is also on the path between Nice and Paris. It is better to find metadata at Lyon than at Strasbourg.

Read protocol (3/3)

Figure **??** shows the metadata tree once the object is relocated at Nice.

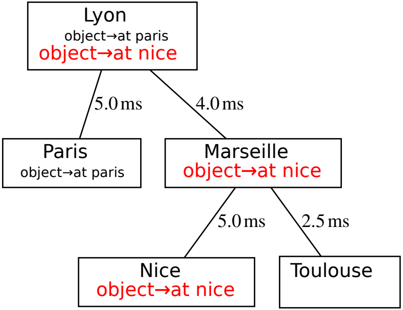


Figure 11: Metadata tree once the object is relocated at Nice.

Experimental Evaluation (1/2)

We measure the time to **locate the objects** in the two approaches (we consider different replication levels in the DHT).

- 1000 objects are written at Strasbourg and are read successively from the others sites (the order is different for each object).
- During a read operation, each object is read one and only time from any site that does not have accessed the object before.
- Test was executed 10 times and average result is presented.

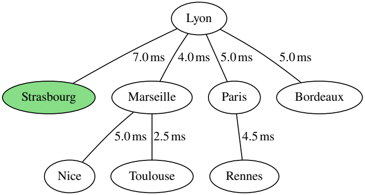
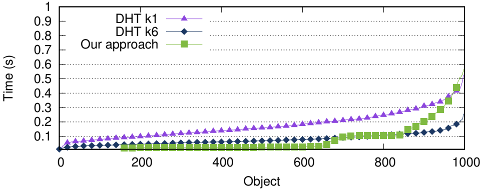
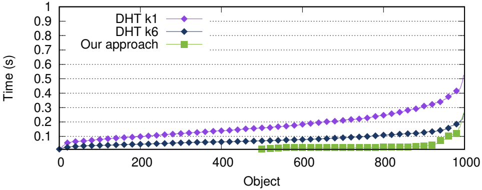


Figure 12: Tree used. Objects are written in Strasbourg and read from other sites.

Experimental Evaluation (2/2)



(a) – First read



(b) – Sixth read

Figure 13: Times to locate the objects in the first and in the sixth read. Objects are sorted by the time to locate them.

Advantages & drawbacks

Our approach has several advantages:

Contains the network traffic: Location is always found on a site along the path to the site storing a replica;

Increases locality: If there is a replica close to the node, the location will be found on a close site too (the more object replica, the more location record replicas);

Disconnected mode: By requesting close node first, we enable the system to work if a group of sites is disconnected from the others.

But also some drawbacks:

Update overhead: The number of update messages is variable and may be important;

Root node can become a bottleneck

A more realistic experiment using FIT and G5K platforms

Experimentation using Grid’5000 and FIT

We want to evaluate the performance of our approach using things at the Edge of the Network instead of emulating them on the Grid’5000 platform.

We propose to deploy a Fog Site on the Grid’5000 testbed and the clients on the FIT platform.

Interconnection difficulties

- No direct IPv4 connection between the two platforms (NAT) - only the public address of frontends are reachable from the other platform;
- No IPv6 support on Grid'5000;
- No locality of the VPN gateway provided by Grid'5000.

We have no other choice than establishing a tunnel between the two platforms.

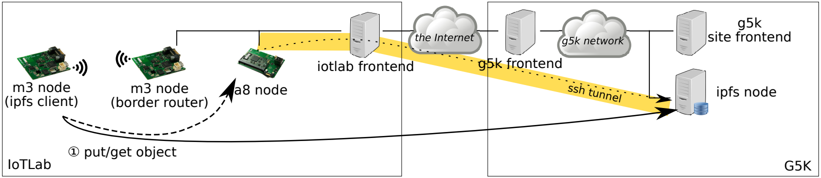


Figure 14: Established tunnel between IoTlab and Grid'5000.

Interconnection difficulties

- The routing between IotLab and Grid'5000 is not optimal (traffic between the two platforms in Grenoble go through Sophia);
- The increase of latency is most important between the A8 node and the M3 node than between the two platforms;
- TCP support in RIOT is still limited, limiting IPFS to objects of 80 bytes!

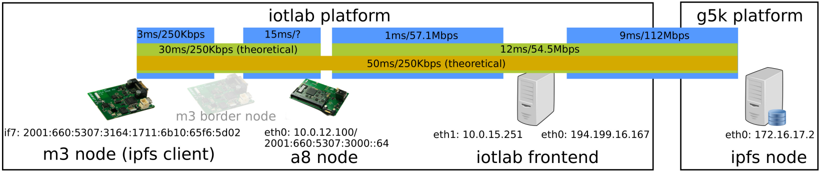


Figure 15: Overhead of the tunnel.

Theses limitations make performance evaluations really difficult to perform.

Results

- We developed a program in RIOT to `put`/`get` an object stored in IPFS.
- No paralellism yet.

Time to write one object from the m3 node: **0.722** seconds (±0.306)

Conclusion

- Coupling a Scale-Out NAS to IPFS limits the inter-sites network traffic and improve locality of local accesses;
- Replacing the DHT by a tree mapped on the physical topology improves locality to find the location of objects;
- Experiments using iotlab and Grid'5000 are not easy to perform.

Questions

bastien.confais@ls2n.fr

RozoFS in a nutshell

RozoFS is a distributed filesystem with the following characteristics:

- POSIX filesystem;
- Metadata server to locate the nodes storing the data;
- Erasure coding (Mojette erasure code);
- Intensive workload: good performance in sequential and random accesses;
- Access through the FUSE API;
- Blocks of **4 KB**, 8 KB or 16 KB.

Read protocol (2bis/2)

Once the object is accessed, a new replica is created locally (at Nice) and location records are created asynchronously. All sites ascendant of Nice in the tree are updated.

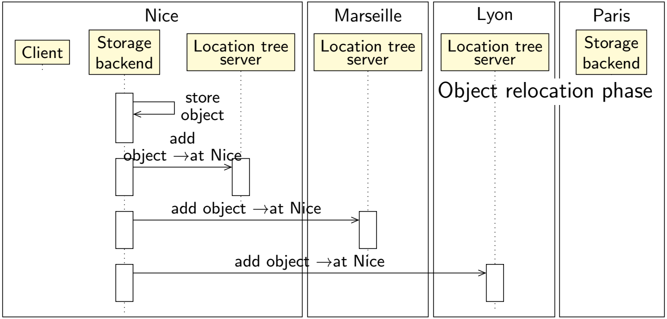


Figure 16: Relocation process when the object stored in Paris is read from Nice.

Interconnection difficulties

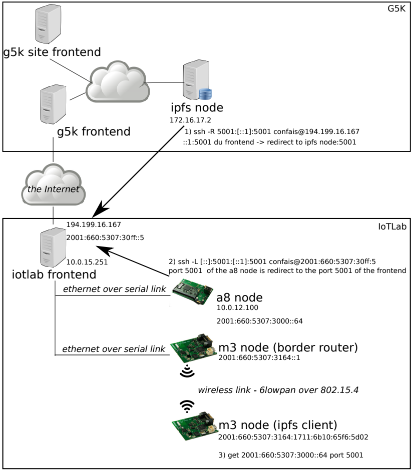

G5K
g5k site frontend
ipfs node
172.16.17.2
g5k frontend
1) ssh -R 5001:[::1]:5001 confais@194.199.16.167
::1:5001 du frontend -> redirect to ipfs node:5001
the Internet
IoTLab
194.199.16.167
2001:660:5307:30ff::5
10.0.15.251
iotlab frontend
2) ssh -L [::]:5001:[::1]:5001 confais@2001:660:5307:30ff:5
port 5001 of the a8 node is redirect to the port 5001 of the frontend
ethernet over serial link
a8 node
10.0.12.100
2001:660:5307:3000::64
ethernet over serial link
m3 node (border router)
2001:660:5307:3164::1
wireless link - 6lowpan over 802.15.4
m3 node (ipfs client)
2001:660:5307:3164:1711:6b10:65f6:5d02
3) get 2001:660:5307:3000::64 port 5001

Figure 17: First connection between iotlab and g5k.

Because, it is not possible to bind a listening port on the iotlab frontend, we establish a second tunnel between the frontend and a A8 node.