A Tree-Based Approach to locate Object Replicas in a Fog Storage Infrastructure

Bastien Confais, Adrien Lebre, Benoît Parrein

A Tree-Based Approach to locate Object Replicas in a Fog Storage Infrastructure

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

Adrien Lebre
INRIA, LS2N, UMR 6004,
Institut Mines Télécom Atlantique,
4 rue Alfred Kastler,
44307 Nantes Cedex 3, France

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

***Abstract*—Fog Computing infrastructures have been proposed as an alternative to Cloud Computing to provide computing with low latency for the Internet of Things (IoT). A few storage systems have been proposed to store data in those infrastructures. Most of them are relying on a Distributed Hash Table (DHT) to store the location of objects which is not efficient because the node storing the location of the data may be placed far away from the object replicas. In this paper, we propose to replace the DHT by a tree-based approach mapping the physical topology. Servers look for the location of an object by requesting successively their ancestors in the tree. Location records are also relocated close to the object replicas not only to limit the network traffic when requesting an object, but also to avoid an overload of the root node. We also propose to modify the Dijkstra's algorithm to compute the tree used. Finally, we evaluate our approach using the object store InterPlanetary FileSystem (IPFS) on Grid'5000 using both a micro experiment with a simple network topology and a macro experiment using the topology of the French National Research and Education Network (RENATER). We show that the time to locate an object in our approach is less than 15 ms on average which is around 20% better than using a DHT.**

I. Introduction

While largely adopted, the Cloud Computing relies on few datacenters located far from the users. This model cannot satisfy the new constraints of the Internet of Things (IoT), especially in terms of latency and reactivity. The Fog Computing approach proposed by Cisco in 2012 [1] consists in deploying from micro to femto datacenters geographically distributed, at the Edge of the network. The proximity to the users enables the Fog infrastructure to provide low latency computing.

Although object stores provide locality and scalability which are two good properties for Fog Computing infrastructures [2], the records containing the location of all objects replicas are often stored without any locality. Most of the time, a Distributed Hash Table (DHT) is used for this purpose [3], [4], [5]. When an object is requested, two sites have to be reached: a first one to determine the object location and a second one to actually retrieve the requested object. Finally, a local replica is created to improve the performance for future accesses and the location record in the DHT is updated to reflect this new location. All objects are replicated where they are accessed but the location record is not relocated, we claim that locating an object may be expensive.

In this paper, we propose a location management approach in which the location of objects is replicated on the fly in a tree. Our approach is inspired by the Domain Name System (DNS) [6] and by some Content Network Delivery [7] protocols. Its main characteristic is to take into account the physical network topology to relocate the location record along the routing path to the object's replicas. To achieve this, our proposal also includes a method to generate a tree from a network topology adapted to Fog environment. Evaluations using the object store InterPlanetary FileSystem (IPFS) [5] on the Grid'5000 testbed [8] shows our approach reduces the time to locate an object by 20% on average.

The remaining of the paper is organised as follows. In Section II, we describe why a DHT is not adapted for Fog infrastructures. We give a technical background to understand our protocol and we present the assumptions we made in our work. In Section III, we describe our protocol and discuss our method to build the tree. Our approach is then evaluated in Section IV, and finally, related works are presented in Section V before concluding in Section VI.

II. Technical background

Writing objects on the closest Fog site is a valuable property for an object store working in a Fog infrastructure in order to reduce the access times [2]. A global DHT spread among all nodes is used to store the relation between each object's name and the location of its replicas. Accessing a DHT to find the location of an object has many drawbacks. Soliciting a remote site to get the location of an object impacts the reading performance because the network latency to reach the site storing the location record may be high. The performance of the remote site is also impacted because answering external queries increases the load of the servers. Finally, it prevents the system from working in case of network partitioning: if a site is isolated from the others, nodes cannot locate the objects stored on it. This approach is used in storage solutions such as IPFS, which is an object store relying on a BitTorrent like protocol to exchange the objects between the nodes and on a Kademlia DHT to store the location of these objects.

Two properties are expected from an efficient location management system in a Fog infrastructure. First, location records should be found close to the site accessing the object

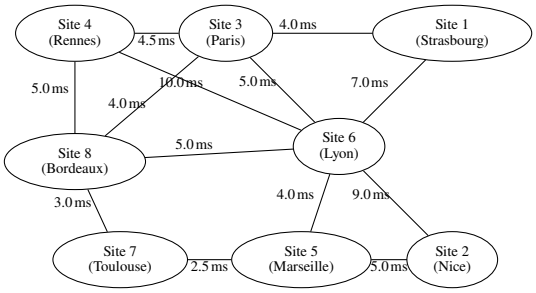


Fig. 1: Part of the French National Research and Education Network physical topology.

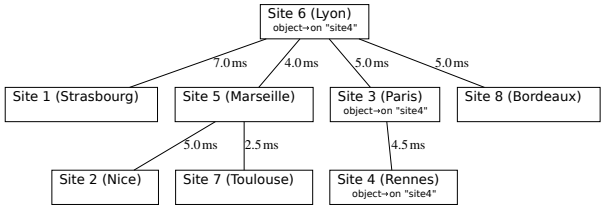


Fig. 2: Tree overlay generated from the French NREN topology, showing the location records of an object stored on Site 4.

and secondly we should benefit from data movements between sites to improve the locality: when a site accesses an object and relocates it, other sites should be able to locate this new replica by reaching a closer site than the original one. We illustrate these properties using Figure 1 showing different points of presence of the French NREN (RENATER) topology but also their physical connections with their network link latencies (see https://renater.fr for more details). We consider this network latency as the distance between the sites. This locality cannot be provided by a DHT where the site storing the location of the object is determined by a hash function.

Because the DHT is not able to provide physical locality and to relocate the records storing the location of object replicas close to where they are needed, we propose a protocol inspired by the Domain Name System (DNS). In the DNS protocol, a resolver who wants to perform a query, first requests a root node and if the root node cannot answer to the query directly, it indicates to the resolver which server is the most able to answer. This mechanism is similar to the hops performed in a DHT but with a major difference: it is possible to choose the node storing the location of object replicas instead of using a hash function. It is also possible to control the servers that a node needs to reach to perform a query.

In addition to managing the location through a hierarchical approach, we build a tree overlay on top of the graph shown in Figure 1. We propose in Section III-B, to use a modified version of the Dijkstra's algorithm that generates the tree shown in Figure 2.

III. A TREE BASED APPROACH FOR OBJECT LOCALISATION

This section first describes how our protocol works, then proposes an algorithm to generate the tree we rely on.

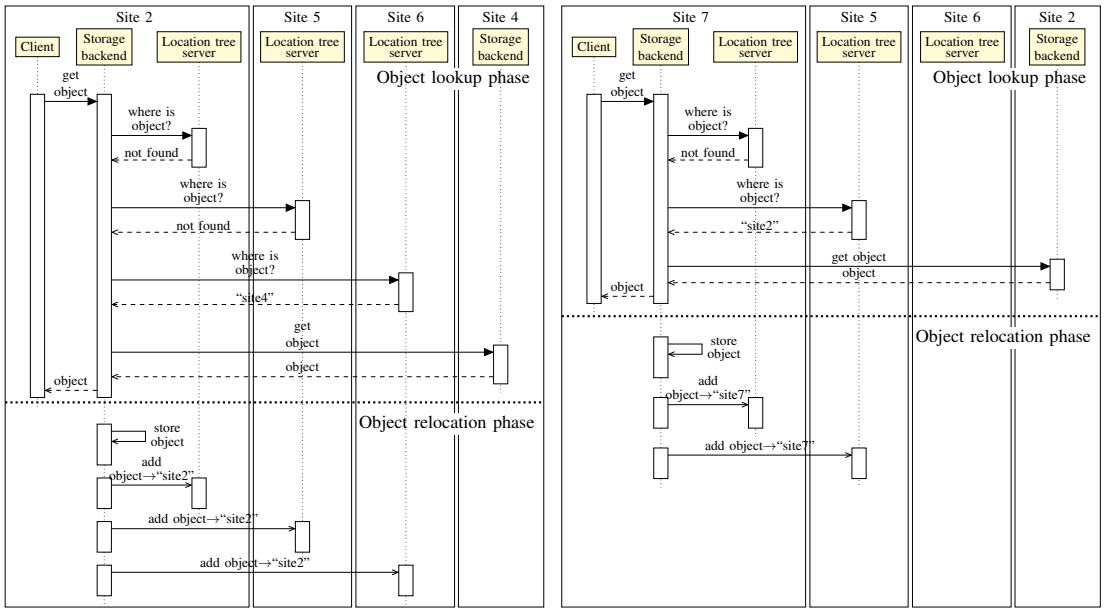
In our work, we make several assumptions. First, we work in a context where objects are immutable and therefore all replicas have the same content and are consistent. Second, the requests are sent in an iterative way. In other words, the node that looks for an object, sends the queries to the different nodes. Although the recursive approach is more efficient, most DHT works in an iterative manner [9]. In this study, we kept this approach for fairness reason even if a recursive approach may be easily implemented in next version of our protocol. Finally, we point out that a request does not lead to know the location of all object replicas but only the closest ones in order to reduce access time at its minimum.

A. Protocol description

We propose to distribute the records storing the location of object replicas inside a tree. Like in the DNS protocol, the different names are spread over different servers organised in a hierarchical way. The tree is composed of the different sites of Fog and is browsed from the current site to the root. If the location of a replica of the object is not found at a given level, the parent node is requested. Contrary to the DNS where a resolver first requests the root node, our protocol uses a bottom-up approach. We consider the tree is organised according to the physical topology including the links latencies in order to minimize the time to find the node storing a replica of the object. In other words, the parent of each node is physically close and looking for the location of the object by requesting it, is faster than requesting the parent of the parent. It also limits the network traffic to a small part of the topology.

Figure 2 also shows the location records organisation. The edges between the nodes correspond to physical network links. Each node is able to answer to all requests for objects stored in their subtree, and more specifically, the root node located in Site 6 is able to provide an answer to all the requests. The root node was chosen because of its central position in the physical topology. We consider each site is composed of a "storage backend" and a "location tree server". The "storage backend" is in charge of storing the objects but also of retrieving them from other sites when it is not stored locally. The "location tree server" is responsible for storing the association between an object's name and all sites on which a replica is stored. Concretely, they store location records composed of an object's name and the address of a storage node storing the replicas for this object. For a given object, a server stores at most one record per replica. Figure 2 also shows an example of location record. For instance, there is two location records for the `object` stored on Site 4: one on Site 4 and one on each site located at an upper level in the tree (Site 6 and Site 3 in the example).

Figure 3(a) shows a first reading process when the object is accessed from Site 2. Because the local storage node does not store a replica of the requested object, the location process is started. The first location tree server to be requested is the local one. Because no location record is stored on it, the storage node requests the parent of this location tree server



(a) – Read from Site 2, the object written on Site 4. (b) – Read from Site 7 the object previously read from Site 2.

Fig. 3: Sequence diagrams of network traffic when a client reads an object first from Site 2 (a) and secondly from Site 7 (b).

(*i.e.*, Site 5). The process continues until one location server can answers. In the worst case, the location is found on the root location server (*i.e.*, Site 6). Once the closest location is found, the object stored on Site 4 is relocated locally and new location records are created asynchronously. The storage node sends a message to the location tree servers from the closest one to the one on which the location was found. In this way, sites that are in the subtree of the new created replica will be able to find it during the next read.

We point out, it could be possible not to use "location records" but to store directly object replicas within the tree, avoiding to first determine the location of an object before actually retrieving a replica. However, this strategy is not efficient in term of space usage.

Figure 3(b) shows that when Site 7 requests the object previously relocated in Site 2. The reading process is the same as previously described but the replica from Site 2 is now accessed thanks to the location record found on Site 5. There is no need to reach and update the root metadata server on Site 6, because Site 7 and Site 2 are sibling nodes.

To conclude on this section, we argue our protocol is more adapted for Fog infrastructures than the DHT because location is found along the physical path from the current node to the root node. Finally, in addition to reducing the lookup latency, creation of location records enables the sites to locate reachable objects replicas in case of network partitioning, increasing Fog sites autonomy.

B. Tree computation

As shown in Section III-A, the total latency to find a location increases with the depth of the tree. Our goal is to create a tree in which object's location can be found in a few low latency hops. The classical algorithm to compute the shortest paths

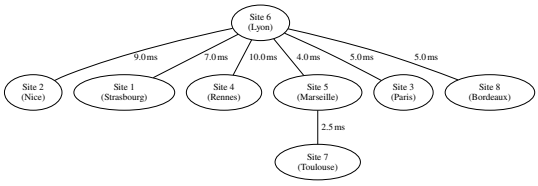
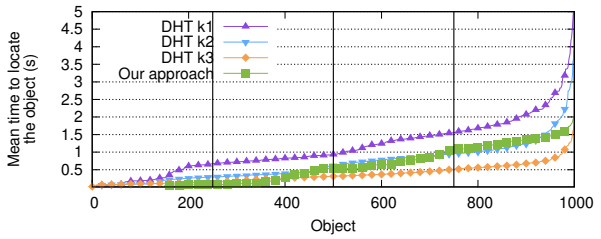


Fig. 4: Intermediate tree generated with our cost function of the Dijkstra's algorithm (total weight: 49). This tree is not adapted to our protocol.

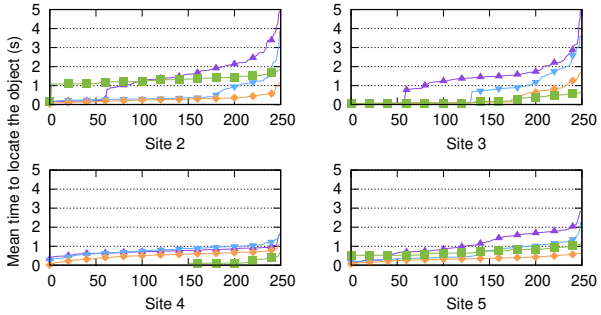
from a source node to all the other nodes is the Dijkstra's algorithm [10]. The generated shortest paths may be seen as a tree with the source node as a root. Therefore, we propose to reuse this algorithm to generate our tree. Nevertheless, some improvements need to be made. The first drawback of the Dijkstra's algorithm is that the root node needs to be specified manually. In order to choose the "best" root node, we compute the tree with each node as source and select the one with the lowest weight. The second drawback is that the traditional cost function does not reflect the way our protocol requests the nodes. Because our protocol is iterative, deeper a link is, more used it is. For instance, in the tree of Figure 2, the distance between Site 7 and Site 6 contains two times the use of the link between Site 7 and Site 5. We propose to use the cost function presented in Equation 1 that reflects this iterative approach.

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

The generated tree shown in Figure 4 is very flat and cannot be used because sites cannot benefit from relocations and access the root node directly. In order to generate a deeper tree, we propose to relax the constraint used in the Dijkstra's



(a) – All sites



(b) – Per site

Fig. 5: Mean times to find the location of objects in the first read for all sites (a) and for each site (b). Objects are sorted by their location time.

algorithm. Instead of selecting a position if it reduces the total weight, we propose to select a position if it does not increase the latency by a factor greater than c in case of the new position of the node is deeper in the tree. Figure 2 shows the tree generated when $c = 1.2$.

IV. Experimental evaluation

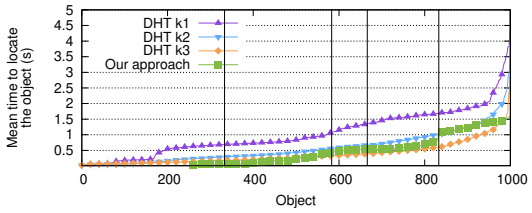
We now compare experimentally our approach to a Kademlia DHT [11] in a micro and a macro benchmark.

A. Material and Method

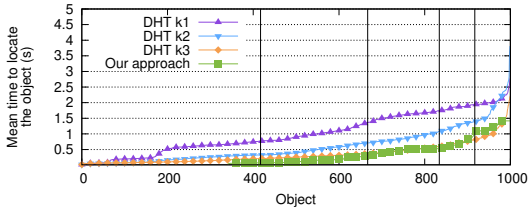
We performed our experiments on the Grid'5000 testbed [8] using the object store IPFS. In order to mitigate our development efforts, we implemented our approach using several DNS servers to store location records. We used the BIND DNS server as a "location tree server" and we modified the routing mechanism used in IPFS to request those DNS servers in a bottom-up manner rather than the DHT.

We use the topologies shown in Figures 7 and 2 for micro and macro benchmark respectively. Objects are written on the first site (in blue in the Figures) and then they are accessed in parallel from other sites. In each read, each object is accessed from one and only one site that did not access it previously. In this way, we never read objects that have been locally stored and for which determining their location is not needed.

Accesses are performed in parallel. We measure the time to access a location record for each object and the number of network links crossed to reach it (the number of hops). Different replication levels in the DHT are evaluated in order to be fair with our approach which relocates location records on the fly. We call "DHT kx", the DHT storing x replicas of each key. The object repository of IPFS and the zone file of



(a) – Second read



(b) – Third read

Fig. 6: Mean times to find the location of objects in the second read for all sites (a) and for each site (b). Objects are sorted by their location time.

the DNS servers are stored in a `tmpfs` in order to prevent any impact from the underlying filesystem.

Network latencies are emulated using the Linux Traffic Control Utility (`tc`). Network bandwidth between the sites is set to 1 Gbps. We use 1 000 objects with a size of 4 KB each. Because we measure only the time to locate the objects, their size does not impact our results. All experiments were performed 10 times. Standard deviations are not represented but are all around 0.01 s.

B. Micro benchmarks

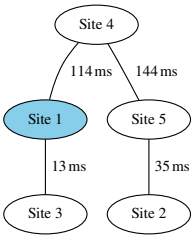


Fig. 7: Tree used for the micro benchmark (total weight: 612). Accessed objects are written on the site in blue.

We perform micro benchmarks on the topology shown in Figure 7 to first validate our experimental plan. This topology is manually built from 5 sites of the Wondernetwork matrix of latencies (see https://wondernetwork.com/pings).

The time to locate objects is shown in Figure 5(a), for both the DHT and our approach when objects are accessed for the first time. Objects are sorted from the time to locate them. It appears that our approach need 1.982 s to locate the 1 000 objects, which is faster than a DHT with only one replica (about 4.794 s) and not so far a DHT with 3 replicas (about 1.740 s). We nevertheless point out that in the first read, our approach relies only one location record. Vertical black lines in Figure 5(a) show the theoretical values delimiting groups of

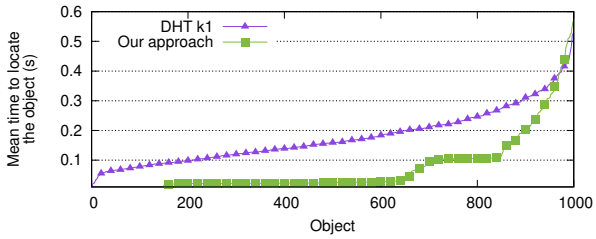
objects for which the location record is reached with the same network latency in our approach. In the first read, because each site locates objects with a different latency, we observe 4 different periods (one every 250 objects) separated by 3 different theoretical thresholds. For instance before the first threshold we mostly observe objects read from Site 4, for which the location of object replicas is stored locally. The second group is almost composed of objects read from Site 3 for which the location of objects is found on Site 1, reachable in 13 ms. From object 500 to 750, we observe objects read from Site 5. Finally, after the last threshold, we observe the objects read from Site 2 which need to reach Site 5 and Site 4 to locate them. Because objects are sorted by the time to locate them and because this time is not constant for each site, theoretical thresholds do not delimit exactly what is happening site by site. To deeply understand the behaviour of each site, we split the Figure 5(a) according to the site requesting each object. Figure 5(b) shows the time to locate an object is not the same for each site because the network latency to connect them is different. For instance, in the DHT, Site 4 cannot reach another node in less than 114 ms because it does not have closer neighbour whereas Site 3 can reach Site 1 in 13 ms. We do not observe non-linearity in our approach for a given site because each of them finds all the location records from the same location. We point out the tail for last objects is due to a bad parallelism of IPFS we checked with sequential accesses (not presented here due to space constraints).

Figure 6 shows that in a second and a third read, the time to locate the objects does not vary with the DHT but decreases in our approach that creates new location records when objects are accessed. For the third read, our approach becomes better than the DHT using 3 replicas: location records are close to the sites which need them instead of being spread uniformly. We need 1.417 s to locate the 1 000 objects whereas the DHT needs 2.206 s in this case. Because new location records are created according to object's access, these theoretical thresholds vary with the different reads. Therefore, for the second read, location is found locally for 333 objects because Site 5 now store the location for objects that have been read from Site 2 in the first read. A similar observation is made on Site 2 for which objects are located from Site 5 instead of Site 4.

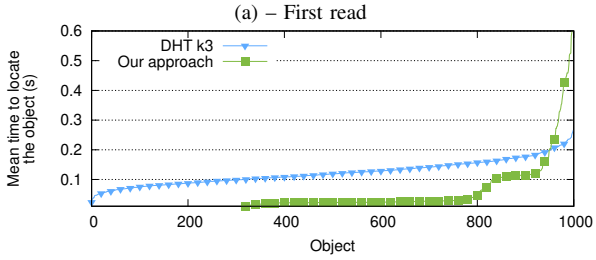
The conclusion of this experiment is that by requesting close nodes first and by creating new location records read after read, our approach reduces significantly the access times.

C. Macro benchmark

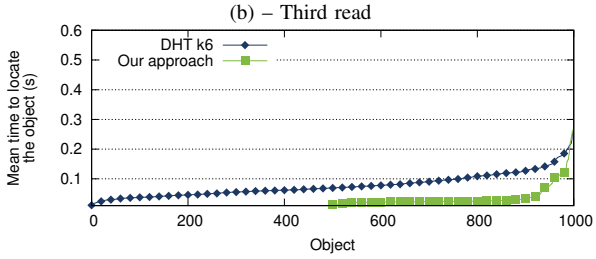
In order to evaluate our approach, we use the original tree in Figure 2 that have been computed using our approach presented in Section III-B. For the DHT-based approach, compute the shortest path (using the traditional Dijkstra's algorithm) in the underlying network, so that each hop has the lowest latency as possible. The consequence is the DHT benefits from optimal network routing paths whereas our approach is constrained by the tree. We perform the same experiment as in the previous section. 1 000 objects are written on Site 1 and



(a) – First read



(b) – Third read



(c) – Sixth read

Fig. 8: Times to locate objects in the first, the third and the sixth read.

are read successively from other sites. Writing on the Site 1 is the worst case possible because for the first access, all sites have to reach the root node.

Figure 8 shows the times to locate the objects in the first read, the third and in the sixth read. Because of the high number of sites, we performed the experiment with the DHT up to 6 replicas for a fair comparison.

We observe that for all reads, our approach has a better performance than the DHT, especially because in our approach, the closest nodes are requested first. Figure 8(a) shows time to locate the objects for the first read. The gap we observe around 600 objects corresponds to the objects from which location record is accessed in 1 hop (accesses from Sites 5, 3 and 8) and objects that are located with 3 physical hops (Sites 2, 4 and 7). In the third read, shown in Figure 8(b), our approach becomes better than the DHT with 3 replicas because of the relocation, but we point out the number of location records is different for each object. For instance an object read from Site 6, then from Site 5 benefits only from 2 replicas when the third read is performed. Contrary to this, an object accessed from Site 2 and then from Site 4 benefits from 5 sites storing at least one location record in the third read. Finally, Figure 8(c) shows better performance in our approach because when location is not stored locally, it is requested on a closer node.

These experiments show that our approach, by creating new location records on the fly, on the path to the root of the tree by taking into account the physical topology enables the nodes to

locate the objects with lower access times than using a DHT.

V. Related works

Many works propose to add locality in DHT but most papers only consider "routing locality". That is, only nodes with an identifier comprised between the identifier of the source node and the identifier of the destination node can be reached during a lookup. This strategy enables nodes to access the closest replica in the identifier space. This strategy is proposed by many DHT using a Plaxton routing [12] to access the closest replica without requesting a node that is further than it. For instance, Zhao *et al.* proposed Tapestry [13], a DHT in which the time to reach a key is proportional to the distance to reach the node storing the key. Yalagandula *et al.* [14] point out that routing locality is not sufficient and they propose a DHT providing path convergence. The advantage is to enable administrative isolation between some sets of nodes, enabling sites to work in case of network partitioning, just like in our Fog Computing approach. As summarized by Castro *et al.* [15], the main drawback of those approaches is that the distance is computed in the nodes identifier space that does not reflect the physical topology. To solve this problem, it is possible either to select the nodes' identifiers or to change the DHT routing tables according to the network topology but these proposals are difficult to implement.

We also mention that comparing a DHT based-approach with the DNS has been already done several times [16], [17]. Pappas *et al.* [17] already showed the DNS approach has a better availability due to the hierarchical caches and also because the average path length to reach the record is shorter. But their experimentation is not fair because they did not use the same number of nodes in the DHT and in the DNS approach. Moreover, this study does not consider physical locality and is not proposed in a Fog Context.

Finally, Content Delivery Networks (CDN) sometimes rely on a tree [18] but contrary to our approach the data are always written at the root, leading to use different trees, depending on the site the object has been written on.

VI. Conclusion and Future work

In this paper, we introduced a new protocol inspired by the DNS to store the location of objects in a Fog infrastructure. We explained how our protocol works, how to compute the tree used, and we evaluated it on the Grid'5000 testbed. The benchmark showed that requesting close nodes first and relocating location records leads to better access times than the DHT. In addition to provide a faster lookup process for objects with a lot of accesses, our protocol also enable the sites to retrieve the closest replica of the objects. Our protocol may benefit several applications like Content Delivery Networks (CDN), P2P approaches like video sharing or the management of virtual machines or container images that all use immutable objects and may be deployed in a Fog environment. Evaluating non-deterministic algorithms to generate our tree and considering a dynamic topology with churn and fault tolerance will be addressed in a future work.

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.

[2] B. Confais, A. Lebre, and B. Parrein, "Performance analysis of object store systems in a Fog/Edge Computing Infrastructures," in *IEEE International Conference on Cloud Computing Technology and Science*, Luxembourg, 2016.

[3] C. Anglano and A. Ferrino, "Using chord for meta-data management in the n3fs distributed file system," in *2004 International Workshop on Hot Topics in Peer-to-Peer Systems*, Oct 2004, pp. 96–101.

[4] M. El Dick, E. Pacitti, and B. Kemme, "Flower-CDN: A hybrid P2P overlay for Efficient Query Processing in CDN," in *12th International Conference on Extending Database Technology (EDBT)*, Saint-Petersbourg, Russia, Mar. 2009, Research Report.

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

[6] P. Mockapetris, "Domain names - concepts and facilities," RFC 1034, Network Working Group, Nov. 1987.

[7] D. Karger, E. Lehman, T. Leighton, R. Panigrahy, M. Levine, and D. Lewin, "Consistent hashing and random trees: Distributed caching protocols for relieving hot spots on the world wide web," in *Proceedings of the Twenty-ninth Annual ACM Symposium on Theory of Computing*, ser. STOC '97. New York, NY, USA: ACM, 1997, pp. 654–663.

[8] 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.

[9] F. Dabek, J. Li, E. Sit, J. Robertson, M. F. Kaashoek, and R. Morris, "Designing a DHT for low latency and high throughput," in *Proceedings of the 1st Conference on Symposium on Networked Systems Design and Implementation - Volume 1*. Berkeley, CA, USA: USENIX Association, 2004, pp. 7–7.

[10] E. W. Dijkstra, "A note on two problems in connexion with graphs," *Numer. Math.*, vol. 1, no. 1, pp. 269–271, Dec. 1959.

[11] 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.

[12] C. G. Plaxton, R. Rajaraman, and A. W. Richa, "Accessing nearby copies of replicated objects in a distributed environment," *Theory of Computing Systems*, vol. 32, no. 3, pp. 241–280, Jun 1999.

[13] B. Y. Zhao, L. Huang, J. Stribling, S. C. Rhea, A. D. Joseph, and J. D. Kubiatowicz, "Tapestry: A resilient global-scale overlay for service deployment," *IEEE J.Sel. A. Commun.*, vol. 22, pp. 41–53, Sep. 2006.

[14] P. Yalagandula and M. Dahlin, "A scalable distributed information management system," *SIGCOMM Computing Comm. Rev.*, 2004.

[15] M. Castro, P. Druschel, Y. C. Hu, and A. Rowstron, *Topology-Aware Routing in Structured Peer-to-Peer Overlay Networks*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2003, pp. 103–107.

[16] R. Cox, A. Muthitacharoen, and R. T. Morris, *Serving DNS Using a Peer-to-Peer Lookup Service*, Berlin, Heidelberg, 2002, pp. 155–165.

[17] V. Pappas, D. Massey, A. Terzis, and L. Zhang, "A comparative study of the dns design with dht-based alternatives," in *Proceedings IEEE INFOCOM 2006. 25TH IEEE International Conference on Computer Communications*, April 2006, pp. 1–13.

[18] A. Benoit, H. Larchevêque, and P. Renaud-Goud, "Optimal algorithms and approximation algorithms for replica placement with distance constraints in tree networks," INRIA, Research Report RR-7750, Sep. 2011.