CAnDoR: Consistency Aware Dynamic data Replication

Etienne Mauffret, Flavien Vernier, Sébastien Monnet

CAnDoR: Consistency Aware Dynamic data Replication

Etienne Mauffret, Flavien Vernier, Sébastien Monnet
LISTIC
Savoie Mont Blanc University
Annecy, France
firstname.lastname@univ-smb.fr

***Abstract*—Data management has become crucial. Distributed applications and users manipulate large amounts of data. More and more distributed data management solutions arise, e.g. Casandra or Cosmos DB. Some of them propose multiple consistency protocols. Thus, for each piece of data, the developer or the user can choose a consistency protocol adapted to his needs. In this paper we explain why taking the consistency protocol into account is important while replication (especially placing) pieces of data; and we propose CAnDoR, an approach that dynamically adapts the replication according to the data usage (read/write frequencies and locations) and the consistency protocol used to manage the piece of data. Our simulations show that using CAnDoR to place and move data copies can improve the global average access latency by up to 40%.**

I. Introduction

With the development of networks and the advent of clouds, data is now shared at geo-scale. Large-scale distributed applications such as social networks, search engines, online stores share a large amount of data. Any user can now use a cloud account to store and share data. Among all these pieces of data, there exist many different manner to use them : several pieces of data are written once and then just read while others are updated frequently; several are accessed by a single user while others are shared among many users.

Depending on the usage and the needs, two different pieces of data do not necessarily need the same consistency protocol. Some of them may require a strong consistency protocol, implying a costly synchronization for each write operation (or for each read operation if the data is frequently updated but rarely read), while others may prefer a more relaxed consistency protocol to reduce access latency and increase the global application performance.

Recent data storage systems such as Casandra [1]and Azure Cosmos DB [2] allows the user to choose, on a per-data basis, the consistency level/protocol that suits the best. There are already many research works concerning the placement of data copies taking into account fault-tolerance concerns or performance concerns [3]–[7]. However, these works do not take into account the consistency protocol used to manage the consistency among the data copies.

We claim that the consistency protocol has to be taken into account while placing (and then moving, while the access patterns evolve) data copies. Of course, the usage pattern remains a key criteria. On the one hand, a read-only data can be replicated at many locations, close to where the data is accessed, to enhance access performance. On the other hand, the number of copies of a frequently updated data should remain small and the copies should remain close to each other to reduce the synchronization cost. However, depending on the consistency protocol and the amount of synchronization among replica it implies (for the write or read operations) the data copies should be placed close to each other (in order to lower the synchronizations latency) or close to the users (in order to lower the accesses latency). Indeed, both the access patterns and the consistency protocol have to be taken into consideration.

The contributions of this paper are the following.

- We describe the interest to take the consistency protocol into consideration while placing data copies.
- We propose CAnDoR, an approach that tracks data accesses and regularly computes, for each data, a replica-set (set of nodes hosting a data copy) taking into account both usage patterns and the consistency protocol used.
- We have written a simulation tool, based on PeerSim [8] providing the ability to simulate a large-scale distributed storage platform and to compute, using CAnDoR, an adapted set of replicas for each piece of data, according to the access patterns and the consistency protocols.

The rest of this paper is structured as follows. Section II presents research works related to data storage systems and data placement strategies. Section III introduces our system model on which our approach, CAnDoR, is built, then Section IV describes how CAnDoR adapts the replication according to both access patterns and consistency protocols. Section V presents and discusses our simulation results before Section VI concludes the paper and proposes some future works.

II. Related works

While designing a large-scale distributed data storage system, data placement is a key problem. This is even more critical for geo-distributed data storage systems. Therefore, there are many research works tackling this issue.

The goals of data placement can be organized into three families: (i) data localization, (ii) data durability (fault toler-

This work was funded by the french ANR project RainbowFS.

ance), and (iii) data access performance (high availability, data freshness).

The data placement for the localization problem has been deeply studied in the context of distributed hash tables (DHT), such as Dhash [9] upon which Chord [10] is built and Past [11] upon which Pastry [12] is built. However, in these systems, data is usually read-only and the replication does not take into account neither consistency, nor user access performance. The main goal here is to offer a scalable way to localize a piece of data and its copies.

Many works have addressed the durability issue. Among them, we can cite Pace et al. [5] which take into account nodes availability while placing data copies or Sun et al. [13] which scatters pieces of data in order to increase the number of potential data sources while healing the system after a failure.

Concerning the performance issue, here again there are many research works. Usually, these works try to place data copies close to the users like Ranganathan and Foster [4]. This is also the case for most content delivery networks (CDN) like Akamai [14]. The works do not take multiple consistency protocols into consideration. In the case of CDNs, the data is read-only.

Some systems propose to adapt the replication during the time [6], [7] (it is also the case for the CDNs). However, even if these systems react, like CAnDoR, to the evolving access patterns, they do not take consistency into account. Scatter [3] takes consistency into account, but its goal is to make linearizability scalable. It does not support multiple consistency protocols.

However, many consistency models and protocols are used by applications. An application may even want to use different consistency models for different pieces of data. A formal description of many models has been proposed by Viotti and Vukolic in [15]. A recent work introduces a new concept and consistency model, such as the "Just-Right" consistency, introduced by Shapiro et al. in [16]. This model provides a low latency response while providing consistency guarantees through the use of an invariant.

Due to the different application needs, several consistency protocols may be implemented in a single distributed storage system. This is already available in commercial systems such as in Casandra [1] and Azure Cosmos DB [2].

For example, Cosmos DB implements 5 different trade-offs between consistency and availability. Each data of the system may have a different consistency level.

Cassandra proposes a different approach of consistency. The number of nodes that must receive an update or the number of nodes that must respond to a request may be adapted to obtain different guarantees over read or write operations. Here again, each data can have its own value for those factors.

In such systems, placing the data without considering the consistency model is inappropriate as each consistency model requires different strategies. However, to our knowledge, no research work focus on placing and moving data copies taking into account both the consistency criterion and the user behavior. CAnDoR aims at filling this gap.

III. System model and assumptions

This section presents the three models we rely on to build CAnDoR. We start by describing the data model then we develop the system model, finally, we present a discussion about the knowledge model. Notice that we do not discuss failures: data-storage node failures have to be tackled by the distributed storage system itself. CAnDoR's algorithms aim at driving the data copies placement on the remaining nodes. The computation itself is fault tolerant: it takes into account only alive (according to the system) nodes.

There is no failure model here because CAnDoR's algorithms should run on top of a distributed storage system and will rely on the failure tolerance provided by this system. If a storage node that has computed some steps in CAnDoR becomes faulty, it will be considered as faulty for the distributed storage system and will be handled directly. This way, the steps already performed are no longer relevant and should not be recovered.

A. Distributed Storage System model

CAnDoR has been designed to place data copies in a distributed storage system. We define such a system by a set of users $U = u_1, ..., u_n$ accessing a set of pieces of data $D = d_1, ..., d_k$ stored by storage nodes $S = s_1, ..., s_m$. Each piece of data is replicated according to a given replication factor, denoted RF, and each piece of data is managed using a particular consistency protocol.

A user may either create, update, read or remove a piece of data. We consider that the users are well formed, i.e. they only perform correct actions. A storage node may either (*i*) serve the request received from a user, (*ii*) share some information with other nodes, (*iii*) move some data or (*iv*) run a CAnDoR computation. We also consider that storage nodes are well formed, i.e. they serve each request once and only once, do not send any message to users apart from responses from requests and communicate with other storage nodes only to share information or to start a data movement.

B. Data model

CAnDoR must be able to run on top of any distributed storage system and therefore does not rely on a given data type. However CAnDoR needs some meta-information concerning the stored data: **(1)** a unique identifier (id), to be able to differentiate data from others, **(2)** the replication factor (RF) and **(3)** the consistency protocol used to manage this data. The consistency protocol is used to determine a couple of coefficients indicating the importance of synchronization (c for Consistency) and response time (a for Availability).We discuss deeper this point in Section IV-B.

Each copy of a piece of data is called a *replica*. If it is not ambiguous, the term *replica* also refers to the storage node that holds a replica.

C. System model

As described above, two sets of nodes are considered: storage nodes (S) and users (U). Each node (storage node or

user) can contact any other node in the system. These sets and the communication properties among nodes are represented by a graph $G = (V, E)$ with $V = U \cup S$ the set of vertices formed by the union of the set of users and the set of storage nodes and E the set of edges such that $e(u, v), (u, v) \in V$ if u and v may communicate together.

We consider that there is no permanent partition in the graph, *i.e.* there is a path from any node to any other node. The latency of a given communication link may fluctuate over time, E is thus enriched by the latency of the communication link depending on time. To this end we use the function $\lambda(e, t)$ that returns the latency of e at time t. As the latency from a node to another may change or as a partition may temporally occurs, we consider the notion of journey J rather than the notion of path. Roughly speaking, a journey $J_{u,v}$ may be seen as a path from u to v through time. This notion is described in detail in [17]. We denoted by $\lambda(J)$ the median duration of the journey J.

The system may be organized in clusters (nodes are partitioned into groups linked by a high speed network) or not.

D. Knowledge model

Storage nodes have an approximate knowledge of the whole system. In case of systems consisting of a federation of clusters, storage nodes may have an accurate knowledge of their own cluster.

A storage node also keeps track on some data-related metrics. For each piece of data it stores, a storage node keeps track of the source and the number of operations send by users, the list of other storage nodes holding a copy of the piece of data, ... Storage nodes of the same piece of data periodically exchange their views of the users behavior.

IV. CAnDoR's approach

CAnDoR is a "per data" approach. A storage node computes the optimal placement for each data independently (each data has its own consistency/performance trade-off and is accessed with a particular pattern). CAnDoR establishes and maintains several metrics to determine, for each data d, which set of nodes should hold a replica.

As each computation considers a single piece of data d, we note $\mathcal{N}$ the set of nodes that holds a replica of d instead of $\mathcal{N}_d$ as it is not ambiguous (and the same is done for other notations).

Roughly speaking, CAnDoR computes the "best" trade-off between the propagation time[1] and response time. These notions are developed in Section IV-A. Section IV-B details how CAnDoR provides the ability to set weights in order to tune the consistency/availability trade-off. Section IV-C explains how the approach adapts the data placement while time evolves, and Section IV-D details the actual computation of the set of nodes.

In order to compute the trade-off between propagation time and response (to the user) time, we need to be able to estimate the time needed for a node u to communicate with another node v. We consider $\mathcal{J}_{u,v}$ the set of potential journeys from u to v. The expected time needed for a message sent by u to reach v is noted $E_{\mathcal{J}}(u, v)$. We assume that this value is provided by the system and may slightly vary over time.

A. Synchronization and response times

When data is replicated in a distributed system, two kinds of guarantees must be provided: the consistency among replicas and a short latency for accessing the data. The CAP theorem ([18]) states that one cannot achieve both strong consistency and high availability (we assume that partition tolerance cannot be avoided in a large-scale distributed storage system). Therefore a distributed system must provide a trade-off between consistency and availability.

However, the system should be as fast as possible, according to the trade-off. Consistency is achieved through some form of synchronization/propagation (blocking synchronization and wait-free propagation are possible according to the consistency guarantees), and availability is measured by the time needed to answer a user request. We develop here how we evaluate those two metrics.

Propagation time t_s: The propagation time t_s is the time needed for a storage node s_i to send an update on the data d to the other nodes that store a copy of d. We denote $\mathcal{N}$ the set of nodes that holds a copy of d. In a fully distributed system, t_s is the time needed for s_i to send the update to the "farthest" node in $\mathcal{N}$. We note $\mathcal{S}(s_i, \mathcal{N})$ this time and we compute it by: $\mathcal{S}(s_i, \mathcal{N}) = \max_{s_j \in \mathcal{N}}[E_{\mathcal{J}}(s_i, s_j)]$. However several synchronization protocols require to send multiple messages (using a consensus protocol like Fast Paxos [19] for instance). We note n_p[2] the number of needed message phases and the formula becomes: $\mathcal{S}(s_i, \mathcal{N}) = n_p . \max_{s_j \in \mathcal{N}}[E_{\mathcal{J}}(s_i, s_j)]$.

If we consider that any node may receive a write operation to propagate, then we must consider the synchronization time of a set that holds a replica of the data d, denoted $t_s(\mathcal{N})$. Such a time is given by the longest synchronization time between any two node of the set: $t_s(\mathcal{N}) = \max_{s_i \in \mathcal{N}}[\mathcal{S}(s_i, \mathcal{N})]$. This metric considers that every node is equally important when considering the synchronization time. One could argue that is not necessarily true. For example, a storage node may have some property (more storage capability, more computation power, etc.). Therefore, CAnDoR has the possibility to handle storage node priorities. The priority given to a storage node is noted p_s. In this paper, however, we consider each storage node as equally important, i.e. $\forall s_i, s_j \in S, p_{s_i} = p_{s_j}$. We finally obtain the following priority-aware formula to estimate the synchronization time of a set:

$$t_s(\mathcal{N}) = \max_{s_i \in \mathcal{N}}[p_{s_i} \mathcal{S}(s_i, \mathcal{N})]$$

Roughly speaking, the synchronization time of a set of nodes is the largest time for two nodes of the set to communicate with each other (enough time to be synchronized), weighted by the importance of storage nodes.

[1] The propagation time is the time necessary to propagate updates to all replicas, it is not necessarily blocking.

[2] The synchronization protocol may vary from a data to another.

Response time to a user t_r^d: If a user sends a request to the system, it will wait for a response from any node, *i.e.*, the fastest one. The waiting time is thus given by the formula: $\mathcal{R}(u, \mathcal{N}) = \min_{s \in \mathcal{N}}[E_{\mathcal{J}}(u, s)]$. If multiple users from a set of users perform read requests (during a given time window), CAnDoR must consider the sum of the response times for all the requests in order to propose the best service for *most* of the users[3]. As for the synchronization time, we provide a priority to each user p_u. A reasonable weight for a user u is to rely on the ratio of read requests made by u over the total number of read requests. However, any other priority distribution can be used depending on the application needs. In this paper we set $p_u = \frac{n_r^{u_i}}{n_r}$ with $n_r^{u_i}, n_r$ the number of read requests performed by u_i and the total number of read requests on d respectively. We obtain the following formula to compute the response time between a set of users $\mathcal{U}$ and a set of storage nodes $\mathcal{N}$:

$$t_r(\mathcal{U}, \mathcal{N}) = \sum_{u_i \in \mathcal{U}} [p_{u_i} \mathcal{R}(u_i, \mathcal{N})]$$

Roughly speaking, the response time between a set of users and a set of storage nodes is the sum of the time for every user to receive at least k responses from the system, weighted by the activity of users. CAnDoR computes, for each data, a set $\mathcal{N}$ of storage nodes such as those two metrics are minimal. However, as explained above, every system must provide a trade-off between consistency and availability. This trade-off may be translated by a different prioritization of those two metrics. Therefore we propose to weight them according to the consistency protocol and the access patterns.

B. Setting the weights: consistency and access patterns

CAnDoR relies on the metrics described in Section IV-A. We explain here how we set the weights in the analytical formulas in order to be adapted to both (*i*) the consistency protocols and (*ii*) the behavior of the users of the system (i.e., data access patterns from a system point of view). Most of the time, if the system guarantees strong consistency properties among the replica of a data, then the system should give more importance to the synchronization time, as this operation is usually blocking the system. At the opposite, a system that provides weak consistency guarantees, such as an eventually consistent or causally consistent system, should prioritize the response time, to be sure to provide fast responses to the users. Of course this also depends on the ratio of blocking operations versus non-blocking operations.

However, depending on the application, these criteria may vary. For instance, if a system provides eventual consistency properties but wants to have some guarantees on the data freshness, it may need to give more importance to the synchronization time. In CAnDoR, these two criteria are modeled by two coefficients: c_c and c_a the coefficients for the importance of consistency and the importance of availability guarantees respectively. For clarity reason, the sum of these two coefficients are normalized, such as $c_c + c_a = 1$. We discuss in more details the values of these coefficients in Section V. Their values are static, fixed at the creation of the piece of data. However, depending on the users behavior, these coefficients are not enough. Indeed, the more the users read a piece of data, the more the response time for this particular piece of data is important, and the more they write on the piece of data, the more the synchronization is important. For those reasons, the weights have to take the data access patterns into account. We thus build weights taking into consideration both: (*i*) the static coefficients (representing the consistency protocol constraints); and (*ii*) the ratios of read and write requests (representing the access patterns constraints). We obtain the two following weights: w_s and w_r for the synchronization time and the response time respectively:

$$\begin{cases} w_s & = c_c \cdot \frac{n_w}{n_r + n_w} \\ w_r & = c_a \cdot \frac{n_r}{n_r + n_w} \end{cases}$$

C. Time adaptation

As CAnDoR offers a method to dynamically adapt the set of storage nodes that hold a replica of a piece of data, it is important to consider a time adaptation. A set that was optimal at some point in the execution may become a bad choice if users change their behavior of location. Therefore, CAnDoR could consider a version of n_r^u, n_w^u that gives more weight to the request done in the last t time units. However, we do not expect the behavior to radically change at each instant, and remembering previous value of n_r^u, n_w^u may give more precision to the computation. Therefore we consider a time-moving window on the historic versions of those value h_w^u, h_r^u that we build in the following way: every time unit, the system divides h_w^u, h_r^u by two and then add to it the value of n_r^u, n_w^u before to re-initialize n_r^u, n_w^u. That way, any request made at some point will impact the computation but will fade with time and old behaviors will eventually be "forgotten"(i.e. they will have a negligible impact on the computation). However if two behaviors alternate with each other, the system will consider both of them. We consider the following weights for CAnDoR's computation:

$$\begin{cases} w_s & = w_\sigma . c_s \cdot \frac{h_w}{h_r + h_w} \\ w_r & = c_r \cdot \frac{h_r}{h_r + h_w} \end{cases}$$

Other uses of n_r^u, n_w^u are also replaced by h_w^u, h_r^u (For instance, the activity of a node described in section IV-A).

D. CAnDoR's Computation

The CAnDoR algorithms use metrics and weights introduced above to compute a set of storage nodes that should hold a replica of a piece of data d. It is expected to run for each single piece of data d. It relies on a set of potential replicas $\widetilde{\mathcal{N}}$. Each set $\mathcal{N}_i \in \widetilde{\mathcal{N}}$ contains exactly RF nodes. If the storage nodes have enough computation power and if the number of potential storage nodes is small enough, $\widetilde{\mathcal{N}}$ is the list of all combinations of RF storage nodes. If the number of potential sets is too high, some heuristic needs to be applied

[3]If it offers a better global performance, a small number of users may suffer higher latency.

to reduce the number of sets in $\widetilde{\mathcal{N}}$. We discuss in more details this point in Section V.

CAnDoR's algorithms compute the target set $\mathcal{N}^* \in \widetilde{\mathcal{N}}$ to hold the replica of d:

$$\mathcal{N}^* = \min_{\mathcal{N}_i \in \widetilde{\mathcal{N}}} [w_s * t_s(\mathcal{N}_i) + w_r.t_r(\mathcal{N}_i, U)]$$

If $\widetilde{\mathcal{N}}$ is the list of all possible sets, then we claim that $\mathcal{N}^*$ is the best placement with high-probability[4] according to the system and the behavior of the user.

Once $\mathcal{N}^*$ has been computed for a piece of data d, CAnDoR needs to determine which nodes in $\mathcal{N}^*$ do not already store a copy of d and what are the nodes in the system that should transfer them a copy. Then, each node that holds a copy of d but is not in $\mathcal{N}^*$ (*i.e.* nodes that were in the previous d replica-set but not in the newly computed one) compute if they should keep d or not. Usually, a high value of w_s should lead the node to remove the piece of data while a high value of w_r will tend to increase the number of copies. The CAnDoR algorithms are expected to run periodically. The period between computations needs to be set to be short enough to be relevant but not too short to not harm the system with a high computation cost.

V. Evaluations

This section presents the preliminary evaluations of our approach. To evaluate the performance of CAnDoR, we have developed a discrete-event based simulator built on top of the PeerSim [8] simulation engine. PeerSim has been designed to simulate large-scale distributed systems. We implement the user nodes and the storage nodes as described in this paper. As CAnDoR's approach is per-data, we simulate only one data in the simulations presented in this paper. However, our simulation tool provides the ability to simulate a distributed system holding many different pieces of data.

A. Metrics

- We evaluate the *performance gain* of our approach.
  - The *response time* for the read and write operations is the main goal of CAnDoR: the placement strategy aims at reducing the average global latency while users access data. We also measure the *contact time*, the time needed to contact the closest storage node for data d. This allows us to evaluate the position of the nearest replica without considering synchronization time (which can be useful to understand CAnDoR's behavior).
  - The data *propagation time* is also an important metric: this time can be part of read or write latency (and thus this metric can also help to understand the system's behavior) while using strong consistency protocols; it can also impact data freshness while using relaxed consistency protocols.
- The *computation time* is also studied as it represents the biggest CAnDoR's overhead.

[4]According to the weights and the metrics introduced above. With high-probability because the node that runs the CAnDoRcomputation may has some stale or inaccurate information about the users behavior then $\mathcal{N}^*$ may not be optimal.

B. Simulations scenario and parameters

In all the simulations, one single data is created and accessed by 50 users. The size of the piece of data is considered as negligible: we therefore use the term *latency* and do not take bandwidth into account.

The 50 users may have one of the two following profiles: *Worker* or *Lazy*. In both case, a user performs a loop (from the beginning of the simulation until the end) in which it: (*i*) performs an action, then (*ii*) waits a given time. For a lazy user an action may be a request with a probability of 0.1 or it can be "do nothing" ($idle$). On the opposite, a worker has 0.9 chance to perform a request. For both lazy users and workers, each request may be either a read or write request, with equal probability. In the last experiment, user switch profile at the middle of the run.

Initially, the storage nodes are chosen uniformly at random. Then we let CAnDoR adapt the placement according to the parameters introduced in this paper.

The storage nodes are organized in clusters, the communication time among storage nodes within a cluster is fixed, set at $1ms$. The latency between two clusters goes from $7ms$ to $19ms$ and it is between $11ms$ and $31ms$ between a cluster and a user node. Therefore, if only one replica is allowed in a cluster, the best propagation time is $7ms$ and the best response time is $20ms$ (the shortest latency between two clusters and a round-trip time between a cluster and a user node respectively). The number and the size of clusters vary from one simulation to another. We can also use a high number of clusters of small size to simulate a very large-scale topology (with many nodes, not necessarily gathered in clusters).

C. Results and analysis

computation time: In the first series of simulations, we evaluate the computation time C needed by CAnDoR to find the best set. As we propose here an exact approach among the potential sets, CAnDoR needs to evaluate each set in relation to each user. If we denote N_{set} the number of sets and N_{user} the number of users, the computation growth linearly with those factor: $C = \mathcal{O}(N_{set} * N_{user})$. To obtain an exact solution, we must consider every combination of RF nodes among every storage nodes. In this situation we obtain a factorial growth of the computation time $C = \mathcal{O}(N_{node}! * N_{user})$ with N_{node} the number of nodes.

We observe, as detailed in Figure 1, that under 130 nodes ($\sim 3 * 10^5$ potential sets) and 50 users the computation lasts less than $1s$ and it lasts more than 1 minute with 535 nodes ($\sim 2.5 * 10^7$ potential sets) and 50 users. To mitigate this time, it is possible to consider only 1 or 2 representative nodes by clusters, if it is appropriate for the system. Indeed, the communication within a cluster is usually much faster than between two clusters or between a cluster and a user node. In such situations, considering only one representative node instead of all the cluster nodes provides the ability to

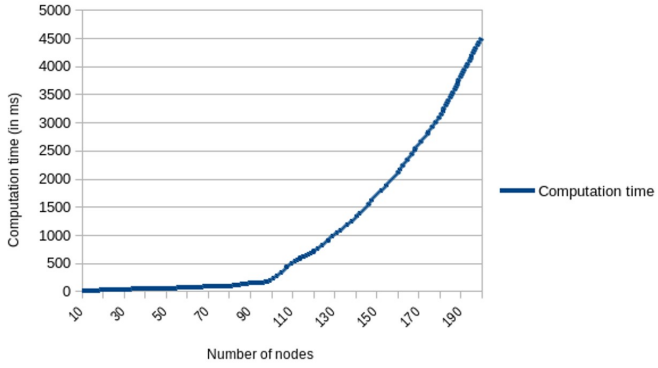

Fig. 1. Computation time according to the number of nodes

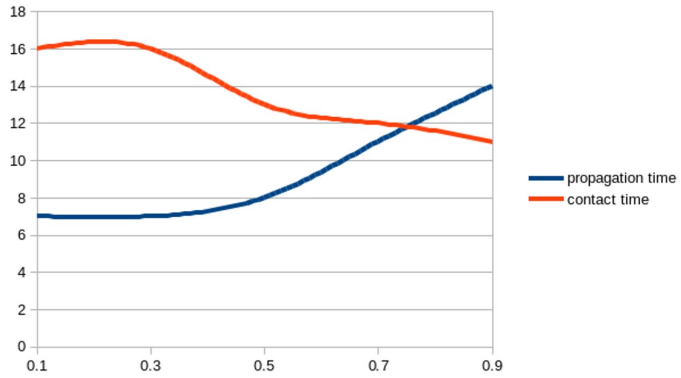

Fig. 2. Computation time according to the number of nodes

drastically reduce the number of potential sets while still guaranteeing a near-optimal result. In future works, we plan to propose heuristics to reduce the number of potential set in large heterogeneous systems (not necessarily following a cluster federation topology but rather arbitrary graphs).

c_a, c_c *coefficients:* In order to determine the best couple (c_a, c_c) we performed a series of simulation with 50 users, where 7 of them are workers, and 60 clusters with 2 representative nodes by clusters. We pose $c_c = 1 - c_a$ and make c_a vary from 0.1 to 0.9. We evaluate the propagation time and the median time needed for any storage node to contact a worker node (Figure 2). It is important to note that this metric differs from the response time as it does not consider the time when the request occurs and therefore any potential delay due to a synchronization. Thus, this metric does not depend on the consistency protocol.

With $c_a \leq 0.3$, we observe that the optimal propagation time ($7ms$) is reached while the time to contact a worker is around $16ms$. With $c_a \geq 0.7$, the propagation time varies from $11ms$ to $18ms$ but provides the ability to contact a worker in $11ms$ in median (from $10ms$ to $13ms$). Finally, when $c_a = c_c = 0.5$ we observe that CAnDoR alternates between two preferred sets: one with good propagation time ($7ms$ or $8ms$) and a medium time to contact a worker, around $14ms$ (from $12ms$ to $15ms$) and one with better time to reach a user ($10ms$ to $13ms$) but a longer propagation time ($9ms$). This behavior is a consequence of the equal distribution of read and write operations and the same importance to the two constraints representing consistency and availability.

From those observations we recommend using parameters close to the following couple in a system that provides strong consistency properties: $(c_a = 0.3; c_c = 0.7)$. This couple provides the ability to get close to an optimal propagation time (to get a synchronization as fast as possible) while still offering a reasonable response time. Obviously this depends on the network behavior and the system parameters. In all the simulations with strongly consistent data we will use these values. In a system with eventual consistency properties, we recommend to users either $(c_a = 0.9; c_c = 0.1)$ or $(c_a = 0.5, c_c = 0.5)$ according to the importance given to the freshness of the data and the number of expected update operations. When using $(c_a = 0.9; c_c = 0.1)$, it is important to note that the propagation time is close to the contact time, which may result in some case of stale data delivered to the users. However we use this couple for the rest of the simulations involving eventually consistent data.

These results have been obtained while restricting the number of replicas to one by cluster. This restriction has been made to avoid data loss at the crash of an entire cluster. If this situation is not expected to happen, it is possible to put several replicas in the same cluster (for load-balancing for instance). In this last scenario, the optimal propagation time becomes $1ms$ and is reached when $c_a \leq 0.3$ and may be reached when $c_a = 0.5$ with a high write operations rate. Therefore one should consider this option only if a whole cluster in not expected to crash and the propagation time is crucial for the application.

Influence of the number of workers and clusters: We run several simulations with different numbers of workers and clusters to measure the impact of those factors on the performance gain of using CAnDoR algorithms. Each of those simulations involves either eventually consistent data or strongly consistent data and are detailed in Figures 3 and 4. We first fixed the number of clusters to 60 with 2 representative storage nodes by cluster and choose the number of workers between 3 and 10.

With a small number of workers (between 3 and 5), CAnDoR provides a great improvement in latency compared to a random placement. With a strongly consistent data, the optimal propagation time is reached while the response time goes from a median of $35ms$ (from $22ms$ to $55ms$) to a median of $30ms$ (from $20ms$ to $35ms$). The high variance in those results is due to the synchronization during writes operations. If no write operation is pending, the response time is equal to twice the latency while the storage node may delay the request for some time in order to finish the synchronization before responding to the requesting user. With an eventually consistent data, the storage node can respond as soon as possible as there is no synchronization. In this scenario the propagation time may reach $8ms$ and the median response time goes from $31ms$

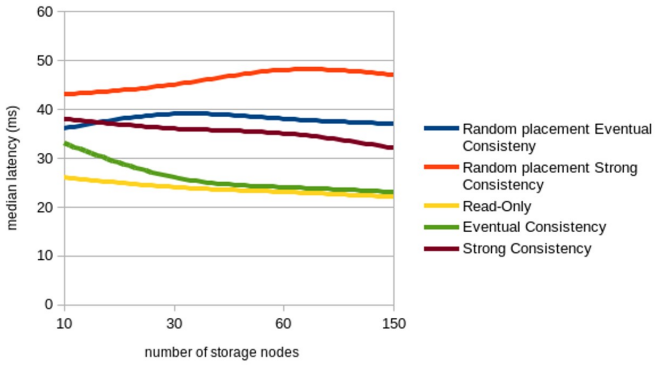

Fig. 3. Computation time according to the number of nodes

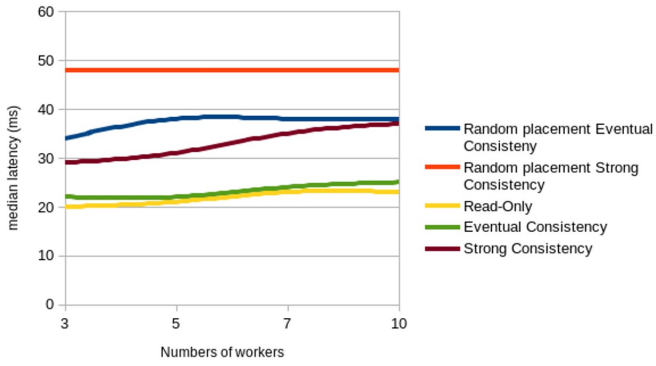

Fig. 4. Computation time according to the number of workers

(from $20ms$ to $42ms$) to $22ms$ (from $20ms$ to $24ms$).

We also perform simulations with similar parameters but in a read-only scenario with an eventually consistent data. The median response time then goes to $20ms$ or $22ms$. As communication times are randomly set at the beginning of simulation, we observed a small amount of scenarios where one of the worker had a high latency with every cluster, which lead to a response time of $32ms$ for this worker.

As the number of worker increases, the gain of CAnDoR slightly decreases, to reach a median response time of $24ms$ (from $20ms$ to $30ms$) for 10 worker with an eventually consistent data and a median of $33ms$ (from $22ms$ to $40ms$) with a strongly consistent data. However the optimal propagation time is almost always reached when using a strongly consistent data. The propagation time for the eventually consistent data is more variable and goes from $8ms$ to $10ms$ depending on the simulation. In the read-only scenario, however, the propagation time goes up to $18ms$ (as there is no write operation and a low consistency constraint) to allow a median response time of $22ms$ with 10 workers.

We then fixed the number of workers to 7 and run simulations with a number of storage nodes from 10 to 150 (here the nodes are scattered: not grouped in clusters). With a small number of scattered storage nodes (between 10 and 20) the gains of using CAnDoR algorithms are small has there is a small variation between each storage node and the random placement may be already close to the optimal more frequently. We note a gain between 2% and 10%. In large systems however, with 150 scattered storage nodes, CAnDoR is much more efficient than a random placement and provides up to 25% of gains with an eventually consistent data (from $30ms$ to $22ms$) and providing a near optimal response time to several worker in the system. With a strongly consistent data, the optimal propagation time is also reached in such system but the response time is bigger: the median contact time is close to the one obtain with a random placement, but the best propagation time allows better performance ($35ms$ instead of $44ms$ for the random placement).

1) Adaptability over time: In order to observe the dynamic adaptability of CAnDoR, we run a scenario in which, at a given time active workers become lazy while lazy users become active workers. There are 10 users, partitioned into 2 groups A and B on 5 users. Here, we show a single particular simulation run (the median and averages are computed on the users in each group).

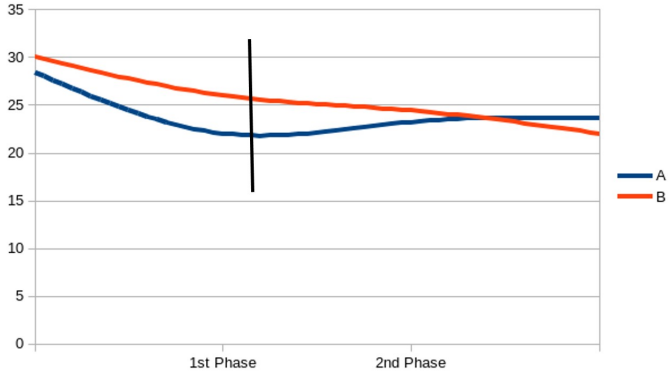

Fig. 5. Access latency evolution during time for users in groups A and B. The black vertical line shows when the users from groups A and B switch profiles.

At first, in phase 1, the 5 users in group A are active workers and then becomes lazy in phase 2. Meanwhile, the 5 users in group B are lazy at first (during phase 1) and becomes active in phase 2.

Figure 5 plots the average latency for nodes in groups A and B with respect to time (the vertical black line materialize the limit between phase 1 and phase 2).

Using a random placement, on this run, the latency would remain close to the initial one: 28 for users in group A and 30 for users in group B. Notice that for this particular run the random placement already offers quite good performance. CAnDoR algorithms enhance the data access latency during phase 1, especially for the nodes in group A (which are more active). Then, while the access pattern evolves, in phase 2, CAnDoR promotes nodes in group B: the placement becomes better for group B nodes while getting slightly worse for group A nodes.

VI. Conclusion and future work

In this paper, we explain how the data placement is crucial in nowadays distributed storage systems and why the consistency protocol should be considered along with the users behavior (access patterns). We then described CAnDoR, our approach to dynamically place and move the data among the storage nodes. CAnDoR keeps track of the consistency protocol of each piece of data and the list of received requests. It uses these metrics to compute a set of storage nodes that minimizes both the propagation time and the response time. The weight of those values is influenced by the consistency model and the access patterns.

We developed a simulator on top of PeerSim to evaluate the performance of CAnDoR. The simulation results show that CAnDoR gives better results compared to a random placement in large-scale systems (up to 40% gains) at a longer computation cost (around $3s$ in a system with 200 nodes). This is due to a larger selection of potential sets. If the number of users increases as well, the median gains of CAnDoR sightly decrease, as CAnDoR determines the barycenter of active workers. However, in several applications, most pieces of data are only accessed by a small set of users (personal data for example).

In future works, we plan to add some tools to handle more complex consistency models, such as the "k-stable causal" consistency and the "Just-Right" consistency models. These models offer different approaches and properties than classic ones. In particular the "k-stable causal" consistency requires to propagate any update operation on k nodes before applying it and to wait for a response from k nodes before delivering a data to the requesting user. To find the best set for such a model we will work on formula adjustment with the notion of "$\min^k$" which give us the k^{th} smallest value of a set.

We also plan to propose some heuristics to reduce the number of sets tested by CAnDoR. That would allow to implement metrics to compute which number of replica is optimal for a particular piece of data, with respect to both the consistency protocol and the user behaviors.

References

[1] Apache Cassandra, "Cassandra," http://cassandra.apache.org/doc/latest/, 2019.

[2] Microsoft AzureDB, "Azure cosmosdb," https://docs.microsoft.com/en-us/azure/cosmos-db/consistency-levels, 2019.

[3] L. Glendenning, I. Beschastnikh, A. Krishnamurthy, and T. Anderson, "Scalable consistency in scatter," in *the 23rd Symposium on Operating Systems Principles*, ser. SOSP '11, New York, NY, USA, 2011, pp. 15–28. [Online]. Available: http://doi.acm.org/10.1145/2043556.2043559

[4] K. Ranganathan and I. Foster, "Identifying dynamic replication strategies for a high-performance data grid," in *Grid ComputingGRID 2001*, 2001, pp. 75–86.

[5] A. Pace, V. Quema, and V. Schiavoni, "Exploiting node connection regularity for dht replication," *Reliable Distributed Systems, IEEE Symposium on*, vol. 0, pp. 111–120, 2011.

[6] W. H. Bell, D. G. Cameron, A. P. Millar, L. Capozza, K. Stockinger, and F. Zini, "Optorsim: A grid simulator for studying dynamic data replication strategies," *The International Journal of High Performance Computing Applications*, vol. 17, no. 4, pp. 403–416, 2003.

[7] O. Wolfson, S. Jajodia, and Y. Huang, "An adaptive data replication algorithm," *ACM Transactions on Database Systems (TODS)*, vol. 22, no. 2, pp. 255–314, 1997.

[8] A. Montresor and M. Jelasity, "Peersim: A scalable p2p simulator," in *2009 IEEE Ninth International Conference on Peer-to-Peer Computing*. IEEE, 2009, pp. 99–100.

[9] F. Dabek, J. Li, E. Sit, J. Robertson, F. F. Kaashoek, and R. Morris, "Designing a DHT for low latency and high throughput," in *the 1st Symposium on Networked Systems Design and Implementation*, San Francisco, CA, USA, March 2004.

[10] I. Stoica, R. Morris, D. Liben-Nowell, D. R. Karger, F. F. Kaashoek, F. Dabek, and H. Balakrishnan, "Chord: a scalable peer-to-peer lookup protocol for internet applications," *IEEE/ACM Trans. Netw.*, vol. 11, no. 1, pp. 17–32, February 2003.

[11] A. I. T. Rowstron and P. Druschel, "Storage management and caching in PAST, a large-scale, persistent peer-to-peer storage utility," in *the 8th ACM symposium on Operating Systems Principles*, December 2001, pp. 188–201.

[12] A. Rowstron and P. Druschel, "Pastry: Scalable, decentralized object location, and routing for large-scale peer-to-peer systems," *Lecture Notes in Computer Science*, vol. 2218, pp. 329–350, 2001.

[13] W. Sun, V. Simon, S. Monnet, P. Robert, and P. Sens, "Analysis of a Stochastic Model of Replication in Large Distributed Storage Systems," *Proceedings of the ACM on Measurement and Analysis of Computing Systems* , vol. 1, no. 1, pp. 1 – 21, Jun. 2017. [Online]. Available: http://hal.univ-smb.fr/hal-01846063

[14] E. Nygren, R. K. Sitaraman, and J. Sun, "The akamai network: A platform for high-performance internet applications," *SIGOPS Oper. Syst. Rev.*, vol. 44, no. 3, pp. 2–19, Aug. 2010. [Online]. Available: http://doi.acm.org/10.1145/1842733.1842736

[15] P. Viotti and M. Vukolić, "Consistency in non-transactional distributed storage systems," *ACM Comput. Surv.*, vol. 49, no. 1, pp. 19:1–19:34, Jun. 2016. [Online]. Available: http://doi.acm.org/10.1145/2926965

[16] M. Shapiro, A. Bieniusa, N. Preguiça, V. Balegas, and C. Meiklejohn, "Just-right consistency: reconciling availability and safety," *arXiv preprint arXiv:1801.06340*, 2018.

[17] A. Casteigts, P. Flocchini, W. Quattrociocchi, and N. Santoro, "Time-varying graphs and dynamic networks," in *International Conference on Ad-Hoc Networks and Wireless*. Springer, 2011, pp. 346–359.

[18] S. Gilbert and N. Lynch, "Brewer's conjecture and the feasibility of consistent, available, partition-tolerant web services," *Acm Sigact News*, vol. 33, no. 2, pp. 51–59, 2002.

[19] L. Lamport, "Fast paxos," *Distributed Computing*, vol. 19, pp. 79–103, October 2006. [Online]. Available: https://www.microsoft.com/en-us/research/publication/fast-paxos/