Large-Scale Causal Data Replication for Stateful Edge Applications

Pedro Fouto, Nuno Preguiça, and João Leitão
NOVA LINCS & NOVA University Lisbon, Portugal
p.fouto@campus.fct.unl.pt, {nuno.preguica,jc.leitao}@fct.unl.pt

2024 IEEE 44th International Conference on Distributed Computing Systems (ICDCS) | 979-8-3503-8605-9/24/$31.00 ©2024 IEEE | DOI: 10.1109/ICDCS60910.2024.00028

***Abstract*—Edge computing is becoming an increasingly popular paradigm, with modern Internet services leveraging hundreds of edge locations to serve their users. However, existing data replication solutions are not designed to operate in this environment, which restricts the edge components of Internet services to operate as read-only caches and entry points for accessing data centers, severely limiting the benefits extracted from the edge.**

This paper presents `Arboreal`, a novel distributed data management system for cloud and edge infrastructures that enables stateful edge applications to be deployed with full (read and write) local access to application data, overcoming the limitations of existing solutions. `Arboreal`'s data replication protocol allows it to automatically and dynamically replicate data across edge locations according to application needs, while providing global causal+ consistency. By relying on a hierarchical topology, `Arboreal` scales to hundreds of edge locations, while recovering from failures in a decentralized and localized manner, without compromising consistency or durability guarantees. Evaluation shows that the scalability of `Arboreal` heavily outperforms state-of-the-art solutions, while the dynamic replication mechanism allows to effectively support a wide variety of edge scenarios including mobile clients.

***Index Terms*—edge computing, causal consistency, data replication, distributed storage**

I. Introduction

The pivotal role of modern Internet services in everyday life leads to the demand of continuous improvements in the response times and availability of these services. By bringing computations closer to clients, *edge computing* addresses these demands in online services, such as social media or online shopping, while enabling novel latency-critical services, such as AR/VR [1], location-based games, autonomous vehicles [2], and live video analytics [3].

Modern Internet services already rely on hundreds of edge nodes to mediate client access to data centers. However, application logic is usually centralized in data centers, since the application data required to handle client requests is traditionally stored in datastores within those same data centers [4]–[6]. This limits the role of edge nodes to serving static content in content delivery networks (CDN) [7], [8] or executing simple computations without manipulating application data in serverless computing [9]. For user requests that must manipulate application data, edge nodes simply operate as reverse proxies, redirecting requests to data centers [10], [11], where they are processed with access to local data storage service replicas.

Deploying application logic on edge nodes has limited benefits if requests require fetching or modifying data that only exists on the cloud. However, extending cloud data storage solutions to allow online services to reap all potential benefits of the edge is non-trivial. Edge nodes often have limited resources, being able to store only a subset of an application's data. This, combined with dynamic client access patterns at each edge node, renders traditional data partitioning techniques unsuitable. Additionally, the significantly larger number of edge locations and their higher susceptibility to failures require a scalable data replication solution capable of handling entire, and frequent, edge location failures. Addressing these challenges to enable fully-fledged applications at the edge requires new data replication solutions tailored specifically for this environment.

This paper presents `Arboreal`, a novel distributed data management system with a decentralized data replication protocol designed specifically to extend cloud-based distributed data management systems to edge environments. `Arboreal` dynamically replicates data across edge locations, adapting to evolving data access patterns, enabling applications to be deployed at the edge with complete local data access, similarly to a native cloud deployment. Simultaneously, it ensures global causal+ data consistency, preventing data anomalies.

`Arboreal` features a decentralized architecture, that leverages on a hierarchical topology where edge nodes communicate directly, eliminating the need for cloud coordination. The benefits of this approach are twofold: (1) enhanced data freshness and efficient handling of mobile clients, and (2) scalability to hundreds of edge locations by preventing metadata and communication costs from growing linearly with the number of edge nodes. `Arboreal` relies on cloud data centers solely for ensuring data persistence, remaining available even if all edge nodes disconnect from the cloud. To overcome edge node resource limitations, `Arboreal` supports dynamic partial replication, creating and removing replicas of data objects based on application needs while enforcing global causal+ consistency. This decentralized process enables nearby edge nodes to replicate data directly, adapting timely to changes in client access patterns and supporting mobility scenarios where clients change their connected edge node. Considering fault tolerance, crucial for the edge due to lower reliability when compared to data centers, `Arboreal` employs a fault-tolerant

This work was supported by Fundação para a Ciência e Tecnologia (FCT) through a Ph.D. scholarship (SFRH/BD/143668/2019) and the NOVA LINCS research unit (UIDB/04516/2020), and by the European Commision through the TaRDIS project (agreement ID 101093006).

2575-8411/24/$31.00 ©2024 IEEE
DOI 10.1109/ICDCS60910.2024.00028

Authorized licensed use limited to: b-on: UNIVERSIDADE NOVA DE LISBOA. Downloaded on August 26,2024 at 17:51:24 UTC from IEEE Xplore. Restrictions apply.

replication protocol with live reconfiguration. This allows clients uninterrupted access to the system in the presence of failures, ensuring consistency and durability guarantees.

To the best of our knowledge, `Arboreal` is the first fully decentralized distributed data management system for the edge with causal consistency. Its replication protocol provides large-scale causal+ consistency, being fault tolerant and supporting dynamic membership, partial replication, and client mobility across different edge locations without compromising consistency guarantees.

Our extensive experimental evaluation confirms that `Arboreal`'s replication protocol scales effectively to hundreds of edge nodes, outperforming state-of-the-art solutions. It also exhibits quick recovery from failures with minimal impact on client latency, ensuring data consistency and durability guarantees. The combination of a hierarchical topology and dynamic replication allows `Arboreal` to adapt efficiently to various edge scenarios, surpassing traditional static replication schemes or centralized solutions. This adaptability accommodates mobile clients and swiftly adjusts to changes on their access patterns.

In summary, this paper makes the following contributions:

- It introduces a system model for stateful edge computing, enabling applications to be deployed at the edge with complete local and consistent data access (Section II).
- It presents the design and implementation of `Arboreal`, a fully decentralized data management system for the edge, ensuring global causal+ consistency and scalability to hundreds of edge locations (Section III).
- An extensive experimental evaluation, demonstrating that `Arboreal` offers substantial advantages in various edge scenarios when compared to alternative solutions, showing both better throughput and visibility times, while efficiently adapting to changing access patterns and mobility of clients (Section V).

II. Towards Stateful Edge Applications

The prevailing deployment model for Internet services involves running main application components in cloud infrastructures, accessing data managed by (potentially geo-replicated) storage systems [7], [11]. In this model, edge locations, encompassing cloud operator points-of-presence, ISP infrastructures, and small regional data centers, handle simple tasks like accessing static content or cached data. To fully leverage the potential benefits of edge computing, providing low latency and reducing centralized component loads, a shift is needed. Adopting a model where edge nodes can fully process all client requests efficiently requires the application logic on edge nodes to have read and write access to local replicas of application data. To support this stateful edge application model, distributed data management systems must extend from data centers to edge locations, addressing the following challenges, distinct from those faced by traditional cloud storage systems [4]–[6]:

Partial and dynamic replication. Applications with large user bases are expected to leverage on a considerable number of edge locations, largely exceeding the typical count of data centers employed nowadays. These edge locations, characterized by less powerful and reliable computational resources compared to core data centers, pose unique challenges, as it is imperative for application components to have unrestricted data access while maintaining consistent guarantees, regardless of executing in the cloud or at the edge. Achieving this requires a data storage solution supporting fine-grained *partial and dynamic replication*. This means that the set of data objects replicated at each edge location evolve over time to reflect the access patterns of clients accessing the applications in that location. This poses a challenge as the replication protocol must dynamically adapt to ensure timely propagation of data updates to the correct locations, preventing components and clients from encountering stale data, and ensuring their operations become visible across the rest of the system.

Scalable consistency. Enforcing some form of consistency across all edge locations is crucial to ensure correctness of applications. While strong consistency simplifies application logic by avoiding data anomalies, it proves impractical for edge settings due to latency and availability issues arising from coordinating numerous replicas. In edge environments, it is more suitable to rely on a weak consistency model, which relaxes consistency for improved availability and response times. To mitigate anomalies in eventual consistency models, many solutions adopt *causal+ consistency* [12], [13] which provides the strongest consistency guarantees while allowing the system to remain available when some replicas are unreachable [14].

Causal+ consistency implicitly captures the *happens-before* [13] relationship, which encodes potential causal relationships between operations. This model ensures that a client never observes the effects of an operation without observing the effects of all operations that causally precede (i.e., that *happened-before*) it. The key challenge lies in tracking these causal relationships. Common approaches use vector clocks [15], [16], but scalability is limited as they grow linearly with the number of replicas and data partitions [15], [17], leading to significant metadata overhead. Other approaches, like tree-based topologies [18], [19], avoid growing metadata costs but are sensitive to changes in the replica set, having weak fault tolerance. Some solutions [20], [21] rely on centralized components, simplifying causality tracking but limiting the benefits of using multiple edge locations. Supporting stateful edge applications requires a scalable solution for tracking causal dependencies across write operations that can *scale to hundreds of locations*, in a context involving dynamically replicated data objects, and in a way that can deal with frequent replica set changes and be fault-tolerant.

Client mobility. Applications benefiting most from the envisioned stateful edge applications often involve numerous users dispersed across different locations, accessing data based on their location. Examples include collaborative applications (e.g. Google Docs), autonomous vehicles, smart-city applications, multiplayer mobile games, or stateful serverless computing [22]. In these applications, where low response times are crucial, users may change locations while using the

Authorized licensed use limited to: b-on: UNIVERSIDADE NOVA DE LISBOA. Downloaded on August 26,2024 at 17:51:24 UTC from IEEE Xplore. Restrictions apply.

application. When doing so, users should be able to migrate from interacting with an application component on one edge location to a closer one seamlessly, without encountering data consistency anomalies. This emphasizes the need for a distributed data storage solution capable of handling *mobile clients* without compromising consistency guarantees.

In the next section, we present the design of `Arboreal` which, to the best of our knowledge, is the first distributed storage system providing *causal+ consistency* while supporting fine-grained *partial and dynamic replication*. It can *scale to hundreds of different locations*, is *fault-tolerant*, and effectively supports *mobile clients*. This unique combination of features makes `Arboreal` especially well-suited for deployment in edge environments.

III. Arboreal Design

`Arboreal` is a distributed data management system designed for the edge, featuring a novel scalable replication protocol that ensures causal+ consistency. Scalable to hundreds of edge locations, `Arboreal` seamlessly adapts to membership changes and effectively manages faults and network partitions. `Arboreal` is designed to support extending cloud applications to edge environments, employing dynamic partial replication. This enables edge nodes to automatically adjust the set of locally replicated data objects in response to changes in client access patterns while allowing clients to move across edge locations without compromising consistency guarantees. All of this is achieved in a fully decentralized manner.

System Model: Our solution assumes a set of cloud data centers spread across different geographic regions, in which a distributed (geo-replicated) NoSQL database is deployed. Deployment specifics of this database (e.g., replication protocol, partitioning scheme) are orthogonal to this work. We consider a set of *edge locations* equipped with computational resources. `Arboreal` extends the database from the cloud data centers to edge locations within each individual region. For this, an instance of `Arboreal` is deployed both in each data center and each edge location. We assume that an edge location may consist of one or multiple edge nodes, however, it is always treated as a single node, with a single instance of `Arboreal` being deployed in each edge location. No assumptions are made about replication and data partitioning schemes within each edge location, focusing instead on data replication across edge locations. To accommodate diverse scenarios and applications, we assume that, at any time, `Arboreal` can be dynamically deployed, along with application components, in new edge locations and that edge locations may fail, or `Arboreal` may be decommissioned from them. Applications deployed on edge nodes rely on `Arboreal` for consistent local data access to support the processing of client operations that can both access and modify application data. Clients, potentially mobile, can migrate between edge locations at any time, and have dynamic workloads, with the set of accessed data objects possibly changing over time.

Data Model: `Arboreal` offers a key-value store interface, akin to other highly available distributed databases [4], [5],

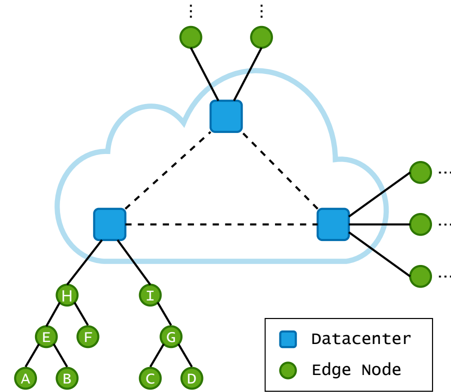


Fig. 1: Design of `Arboreal`, with 3 geographic regions

where each data object is identified by a unique key. Clients issue read or write operations on data objects without constraints, and `Arboreal` ensures that: (1) data objects are replicated transparently to the edge locations where they are accessed; (2) write operations are propagated to all locations currently replicating the modified data object; and (3) clients always observe a state respecting a causal order of operations. `Arboreal` makes no assumptions on how data is stored in each node. For convergence, a *last-writer-wins* policy is used, relying on operation timestamps (explained in III-A2).

A. Replication Model

As discussed earlier, a key challenge of this work is overcoming limitations in replication protocols providing causal+ consistency, in a way that is suitable for a large-scale edge environment. To address this challenge, it is crucial to enable edge locations to synchronize (propagate write operations and data objects) directly with each other while simultaneously ensuring that metadata, essential for enforcing causality and supporting (object-grained) dynamic replication, does not grow linearly with the number of edge locations.

1) Hierarchical Approach: `Arboreal` employs a hierarchical design, with each edge location hosting an instance of `Arboreal`. These edge locations form a tree structure rooted at their regional data center, establishing the region's *control tree*. Fig. 1 shows a geo-distributed example of an `Arboreal` deployment with 3 data centers, each having its *control tree* composed of the edge locations of that region. The management of the *control trees* is fully decentralized, with nodes communicating solely with their parent and children. Nodes only retain detailed information about their children and minimal information about their ancestors (i.e., nodes in the path between itself and the root of the *control tree*). This decentralized structure allows for latency-sensitive tasks, such as creating replicas of data objects (III-A3), failure recovery (III-B2), and mobile client handling (III-C) to be performed in a localized fashion, involving as few nodes as possible.

To establish the *control tree*, when an instance of `Arboreal` is deployed on an edge location, it uses a heuristic to connect to the most suitable existing instance in the *control*

Authorized licensed use limited to: b-on: UNIVERSIDADE NOVA DE LISBOA. Downloaded on August 26,2024 at 17:51:24 UTC from IEEE Xplore. Restrictions apply.

tree of its region. Note that the goal of Arboreal is to replicate data to applications running on edge locations, and is not an orchestrator that decides where and when to deploy the application. To accommodate diverse edge scenarios, both the heuristic defining the *control tree* and the information used by the heuristic are configurable by the application developer.

Given the significance of geographic locality in edge computing, our Arboreal implementation employs geographic distance between edge locations as the primary metric for forming the *control tree*. However, depending on the application, various metrics can be used, such as latency between edge nodes, client locations, or even predicting future demand for the application. Sec. IV provides an insight into how the *control tree* is formed in our implementation.

2) Enforcing causality: The main challenge in providing causal+ consistency in a large-scale edge environment is tracking and enforcing causal dependencies without incurring in prohibitive metadata or communication costs. For this, the key is to leverage the hierarchical topology both to disseminate operations and to aggregate metadata.

Causal Dissemination: We start by leveraging the hierarchical topology of Arboreal, which allows achieving causal consistency without requiring any metadata [18], [23]. For this, nodes form the *control tree* by establishing FIFO links to their parent and children. When a node receives a write operation from a link (i.e., from a parent or child), it atomically executes the operation locally and puts it on the outgoing queue of every other link. Additionally, local operations from clients are atomically added to the outgoing queues of all links. This ensures operations are always propagated (and thus, executed) after all their causal dependencies. While ensuring causal consistency, this approach assumes clients always issue operations to the same node and a static *control tree*, which are unrealistic assumptions for the edge.

Timestamping: To overcome these limitations, Arboreal additionally relies on Hybrid Logical Clocks (HLCs) [24] to enforce causal consistency. HLCs combine physical time for monotonic advancement in each node with logical clocks for capturing causal relationships between operations despite physical clock anomalies. When a client's write operation is received by a node, it is tagged with a timestamp from the local HLC. This timestamp is propagated with the operation through the tree and is stored with the object data in each node. Additionally, Arboreal employs the notion of *Branch Stable Time* (BST). A BST is computed individually by each node as the minimum between its current HLC time and the BST of each child. The BST captures that no operation with a lower timestamp will be generated by any node in its branch (a branch consists in the node itself and all of its descendants in the *control tree*). Nodes periodically propagate their BST to parents and children, including the BST of all ancestors when propagating to children. This ensures each node tracks the BST of its children and all ancestors. Combined with the *causal dissemination* technique, BSTs allow Arboreal to provide causal consistency in every scenario, including failure recovery (discussed in Sec. III-B) and mobile clients (discussed in Sec. III-C).

We note that both the causal dissemination and the timestamping mechanisms require the hierarchical tree topology to function correctly. For the causal dissemination, the acyclic nature of the tree is key to ensure operations are always propagated after their causal dependencies, as using other topologies would require additional metadata (e.g., vector clocks). For the timestamping mechanism, the BST is directly tied to the tree structure, as it represents a lower bound on the future timestamps that can be generated in an entire branch of the tree. Using alternative topologies, such as unstructured peer-to-peer networks, would incur in significant communication and metadata overhead to ensure causal consistency [23].

3) Dynamic and Partial Data Replication: An essential aspect of edge computing is that edge locations can not be expected to have resources to replicate the entire dataset of an application. As such, partial replication is a key aspect of Arboreal. Moreover, to support a wide range of applications, Arboreal must adapt not only to changes in client access patterns but also to mobile clients that can change the edge location to which they are connected at any time. Unlike cloud-based data management systems that typically use static data partitions, Arboreal needs to allow edge nodes to dynamically change the set of replicated data objects at any time with fine granularity. However, keeping track of which nodes replicate which data objects across a large-scale system can be costly and require substantial metadata propagation, especially with dynamic sets of nodes and data objects. To address this challenge, we rely on the hierarchical topology.

In Arboreal, each data object is individually replicated to a subset of edge nodes. This is done in a way that ensures any node always contains the data objects its children replicate, resulting in each data object being replicated across a subtree of the *control tree*, which we refer to as the *replication tree* of an object. Fig. 2 illustrates the evolution of a Arboreal deployment with two objects, α and β, replicated on different sets of edge nodes that change over time.

While this restriction in data object replication may seem limiting, forcing edge nodes to replicate data objects that their current clients may not be interested in, it is actually a beneficial design decision for three reasons: (1) as a subtree of the *control tree*, an object's *replication tree* inherits causal dissemination guarantees, providing causal+ consistency globally across all objects; (2) when the *control tree* is repaired after node failures (detailed in III-B), the *replication trees* involving the faulty nodes are also repaired, enabling the system to quickly recover from failures with minimal client impact; and (3) when mobile clients move between edge nodes, even if their new node does not replicate the required data objects, there is a high chance that one of its nearby ancestors does, allowing the client to quickly resume its operation (detailed in III-C). This design ensures *replication trees* form in a decentralized manner based on client needs. Each node only needs to track the objects it replicates and the objects each of its children replicates. This mechanism assumes that storage and computation capacity increases moving up the *control*

Authorized licensed use limited to: b-on: UNIVERSIDADE NOVA DE LISBOA. Downloaded on August 26,2024 at 17:51:24 UTC from IEEE Xplore. Restrictions apply.

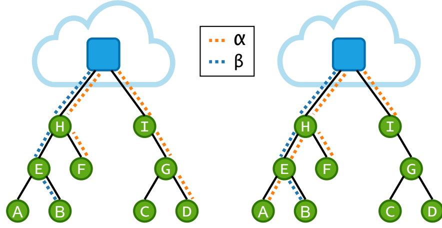


Fig. 2: Dynamic partial replication in Arboreal

tree, closer to cloud data centers, which we believe to be a reasonable assumption for an edge environment [25].

Replica creation: When a client requests a data object not replicated in its connected edge node, that node sends a request to its parent node, asking to be added to the object's *replication tree*. If the parent node is part of it, it sends the current object version to the child node and keeps track that this child node is now replicating the object. If the parent node is not part of the *replication tree*, it forwards the request to its own parent, and so on, until the request reaches a node replicating the object (or the cloud data center, replicating all objects). This node then sends the object to its child node, and the process repeats until the object reaches the node that initially requested it. This process is illustrated in Fig. 2, where node A (and consequently, node E) is added to the *replication tree* of α.

Garbage Collection: Due to the possible large number of data objects and limited storage capacity in edge nodes, Arboreal employs a *garbage collection* process. Each node tracks the last time each data object was accessed and periodically removes objects not accessed for a configurable duration. When removing a data object, a node informs its parent that it no longer replicates that object. Nodes can only garbage collect objects not replicated to any children to prevent breaking the *replication tree*. Fig. 2 shows an example of this, where object α is no longer being accessed by clients in nodes D and G, and so it is garbage collected, with those nodes no longer being connected to the *replication tree* of object α at the end of the process.

The decentralized design of Arboreal's replication protocol allows it to scale to a large number of edge nodes while supporting fine-grained replication of data objects. This allows each node to only track metadata proportional to the number of objects it replicates, avoiding linear growth with the total number of edge nodes and data partitions/objects.

B. Fault Tolerance

Unlike cloud environments, where individual nodes can fail but it is unlikely that an entire data center does, in edge environments, we need to assume that entire edge locations can fail or become partitioned at any time. Therefore, Arboreal must not only be capable of recovering from failures but also provide data persistence guarantees when they occur.

1) Data Persistence: Arboreal provides a mechanism allowing applications using it to specify the *persistence level* of write operations. Effectively, this allows applications to specify how many nodes upstream in the *replication tree* a write operation must reach before it is considered to be persisted and hence having a reply being returned to the application. This mechanism is especially beneficial for applications requiring data persistence guarantees in more volatile edge locations. Its design prevents scalability issues by avoiding extra communication steps between nodes and the need for nodes to track the origin of each operation.

Persistence ID: Before forwarding a local write operation to its parent, a node assigns a *persistence ID* to the operation. Upon receiving a write operation from a child, a node assigns its own *persistence ID* to the operation, mapping it to the child's *persistence ID*, and then propagates the operation to its parent. The *persistence ID* is essentially a local counter for each node, incremented whenever a node assigns it to an operation. The left side of Fig. 3 illustrates this mechanism in action. A client issued three write operations in node A, with the first two reaching the data center, and the last one only reaching node E. Additionally, an operation in node F reached the data center. The figure depicts the mappings by each intermediate node. For example, the data center operation with *persistence ID* $H3$ originated in node A with *persistence ID* $A2$. Importantly, nodes lack information about the origin of each operation, enabling Arboreal to scale by avoiding storing metadata concerning a large number of nodes.

Persistence level notifications: Periodically, each node communicates to its children the persistence level of their operations through a list of pairs *(persistence ID, persistenceLevel)*. Each pair signifies that all child operations up to *persistence ID* have been persisted in *persistenceLevel* nodes. Upon receiving this list from its parent, a node maps the *persistence ID* of each pair to the *persistence ID* of the child, increments the *persistenceLevel* of each pair by one, and, if operations from the child are missing, adds an entry with the highest of those operations and a *persistenceLevel* of 1. The *persistenceLevel* of operations reaching the data center is conveyed as ∞. This mechanism is depicted in Fig. 3, where node A receives acknowledgment that its operations up to $A2$ have been persisted in the data center and $A3$ in one level above it. This persistence level information is periodically sent to children piggybacked on the BST messages. Upon receiving confirmation that an operation has been persisted in the data center, nodes can forget all persistence information related to that operation. Regardless of the requested persistence level, this mechanism is always active for all operations to ensure no loss of operations during fault recovery.

2) Fault Handling and Recovery: Due to the nature of edge environments, decentralized fault handling and recovery are essential for Arboreal. The main challenges involve: (1) rebuilding the *control tree* after node failures; (2) ensuring consistency during the rebuilding process; and (3) maintaining data persistence through failures and reconfiguration.

To address the first challenge, each node, being aware of all its ancestors, can independently rebuild the *control tree* after failures. Nodes attempt to connect to their grandparents if they suspect their parents have fail, falling back to the great-

Authorized licensed use limited to: b-on: UNIVERSIDADE NOVA DE LISBOA. Downloaded on August 26,2024 at 17:51:24 UTC from IEEE Xplore. Restrictions apply.

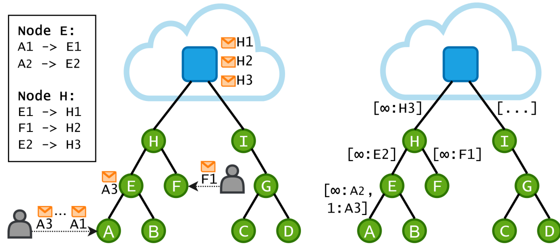


Fig. 3: Data persistence in `Arboreal`

grandparent and so on, until reaching the data center. Since every ancestor of a node replicates a superset of its data (as seen in Fig. 2), this simple approach automatically repairs not only the *control tree*, but also any disconnected *replication trees*. We note that, as we assume an asynchronous system, suspecting a node does not necessarily mean that the node has failed, but rather that there is a chance it might have, as it can just be temporarily slow. However, since a single failed (or just slow) node can block operations from being propagated through the *replication trees*, this mechanism minimizes the impact of such nodes, as when their children change parents, they become leaf nodes that cannot negatively affect other nodes. Simultaneously, this same mechanism can be used to avoid nodes in higher levels (i.e., closer to the root) of the *control tree* from being becoming a bottleneck if they become overloaded due to a large number of children, as in such cases, (some) children can disconnect and reconnect to an ancestor.

For the second challenge, when connecting to a new parent, `Arboreal` employs a 3-step protocol to synchronize with its new parent, ensuring no violations of consistency guarantees:
(1) The (to-be) child node sends a *Sync Request* to the (to-be) parent node, including its current BST and a list of replicated objects with associated timestamps;
(2) The parent node registers the child as a new child, checks if it has any outdated objects, and replies with a *Sync Response* containing the child's new ancestor list, BSTs, and a list of updates for the child's outdated objects;
(3) The child updates its ancestor list and BSTs and installs the outdated objects. It sends the parent requests for any pending replica creation requests and client write operations. Finally, the child propagates a *Reconfiguration Message* to its children, containing their new ancestor list and BSTs, which is propagated to its entire branch.

To address the third challenge, after the synchronization, both the child and every node in its branch re-propagate local write operations with pending persistence requests, as the persistence mechanism may break down during reconfiguration.

Though the synchronization protocol may seem complex, it allows each node to reconnect itself to the *control tree* without centralized coordination. This decentralized approach enables `Arboreal` to recover from multiple failures in parallel. We study the benefits of this mechanism in Sec. V-C.

Regarding clients, failures can affect them in two ways:
(1) If a client's connected node remains operational, but one of its ancestors fails, the client can continue normal operation. The only noticeable effect is that persistence confirmations may be delayed as they will only arrive once the current node reconnects to the *control tree*;
(2) If the node to which the client is connected fails, the client must reconnect to a new node (detailed in Sec. III-C). Operations with persistence confirmation are guaranteed to be visible in the new node, but those without *may* have been lost, affecting the causal session of the client[1]. The client can then re-execute lost operations or perform read operations to verify the persistence of those operations.

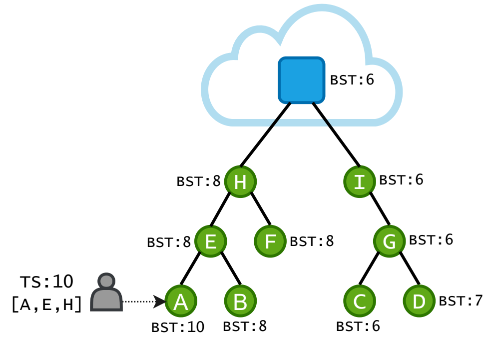


Fig. 4: Client Mobility in `Arboreal`

C. Client Mobility

In edge application scenarios with mobile clients, such as users with smartphones, `Arboreal` must seamlessly support clients moving from an edge node to a closer one while maintaining consistency guarantees (e.g., to support mobile AR applications [1]). This is a non-trivial task as the new node may not have any information about the client's previous node (which may even have failed). Therefore, we assume that the nodes involved in this procedure are unable to communicate, with the client storing all required information.

Client state: Clients of `Arboreal` track two pieces of metadata: (1) the list of ancestors of its current node; (2) a timestamp with the its current causal dependencies. This timestamp is updated upon receiving responses to operations, and always contains the highest timestamp seen. Read operations return the timestamp of the object read, while write operations return the timestamp assigned to them by the client's node. Fig. 4 shows the BST of nodes and the timestamp and list of ancestors of a client in an example deployment.

Mobility procedure: The combination of the client-side timestamp and list of ancestors with the node-side BST enables `Arboreal` to support quick client migrations without compromising causality. When a client connects to a new node, it sends a migration request to the new node, containing its current timestamp and list of ancestors. The new node then compares the received list with its own list of ancestors, leading to one of the following cases:

[1]We note that this case is not unique to `Arboreal`, as in any system with causal consistency, the failure of the node to which a client is connected to will possibly result in the client losing its causal session.

Authorized licensed use limited to: b-on: UNIVERSIDADE NOVA DE LISBOA. Downloaded on August 26,2024 at 17:51:24 UTC from IEEE Xplore. Restrictions apply.

(1) If the new node was not in the client's list of ancestors, signifying a *horizontal migration* to a new branch of the *control tree*, the new node identifies its closest ancestor that is also in the client's list of ancestors. It responds to the client only after receiving a BST from that ancestor greater than the client's timestamp. This ensures that the new node responds to the client only when it is certain that it has observed all operations the client depends on. Using Fig. 4 as an illustrative example, if the client wishes to migrate from node A to node F, then it will need to wait until the BST of node H (the closest common ancestor) reaches a value of at least 10 (the client's timestamp) and that BST is propagated to node F.
(2) If the new node was in the client's list of ancestors, indicating a *vertical migration*, the new node identifies which of its children is an ancestor of the client's old node (or the node itself). It responds to the client once it receives a BST from that child that is equal or greater than the client's timestamp. In Fig. 4, if the client migrates to E, it waits until E receives a BST from A with at least 10 (which should be quick, as the BST of A is already 10). In cases where the client migrates to the parent of a failed node, the new node immediately accepts the migration as there is nothing to wait.

After this process, the client receives an updated list of ancestors. The metadata stored in the client only needs to be readable by `Arboreal` and may be opaque (e.g., encrypted) to the client itself, preventing information leakage about the internal organization of the system.

The duration of the migration process increases as the client moves farther from its old node, requiring the new node to wait for a BST from a more distant node in the *control tree*. However, in typical scenarios, we expect clients to mostly migrate to close-by nodes, resulting in swift migrations. We evaluate this in V-E.

This mechanism might be overly cautious. For instance, in Fig. 4, if the client aims to migrate to F, it depends on the BST of H, which, in turn, relies on the BST of B. However, node B might not have participated in any operation observed by the client, causing potential delays in migration completion. While we recognize this cautious approach may introduce unnecessary delays, alternatives that accelerate migrations typically involve additional metadata or explicit migration messages sent through the tree. Such approaches could compromise `Arboreal`'s scalability and fault tolerance.

IV. Implementation

Our `Arboreal` prototype, supporting all features outlined in Sec. III, is implemented in around 4000 LOC of Kotlin, leveraging on Netty [26] for network communication. It is open-source and extensible, comprising four modules: (1) the main `Arboreal` module manages the *control tree* and *replication trees*, handling message propagation and all other aspects detailed in III; (2) the storage module allows the use of different storage backends. For our evaluation, we implemented an in-memory key-value store; (3) the client module allows clients to interact with the system; and (4) the management module allows nodes to collect information about each other, determining the initial *control tree* topology. Modules communicate asynchronously through message passing.

The management module, while somewhat beyond the scope of this paper, is crucial for our experimental evaluation, as it controls the lifecycle of edge nodes. In line with decentralization, nodes establish an overlay network using HyParView [27], with the datacenter node as a contact point. Upon entry, nodes utilize the overlay to propagate their information, including geographic location, while simultaneously collecting information about others. Using this data, the management module of each node employs a configurable heuristic to select an optimal parent, shaping the *control tree*.

In our implementation, we employ two heuristics to shape the *control tree* layout, both using the following equation: $cost = dist(self, candidate) + w \times dist(candidate, dc)$ which assigns a $cost$ to each possible parent node $candidate$, based on its geographical distance $dist()$ to the node $self$ and the data center dc. This equation prioritizes parent proximity while also considering their distance to the data center, with a weight w. We used the value $w = 0.75$ in our experiments, as it resulted in the most balanced trees.

The first heuristic simply selects the node with the lowest $cost$ as the parent, favoring nearby nodes which are in the general direction of the data center. This results in *control trees* with deep branches and a small number of children per node. The second heuristic limits the depth of the tree by assigning a *level* to nodes based on their distance to the data center. It only considers nodes with a level closer to the data center as potential parents, leading to wider layouts with a larger number of children per node. The impact of these tree layouts, referred to as *deep* and *wide*, is evaluated in the following section.

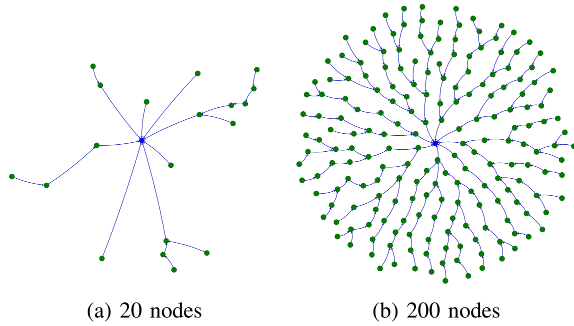
Along with the fully working prototype, we implemented a client driver in Java. Extending the codebase of YCSB [28], it allows evaluating `Arboreal` in a variety of scenarios, allowing clients to move between edge nodes due to node failures or to capture client mobility, select diverse persistence levels, and dynamically modify their workload, adapting to varying data access patterns.

V. Evaluation

In this section, we evaluate the benefits of `Arboreal` using an edge environment setup (Sec. V-A). We start by analyzing system performance, focusing on throughput and operation visibility times (Sec. V-B). We follow by testing `Arboreal`'s resilience to failures, highlighting the role of its persistence mechanism (Sec. V-C). We then explore more complex edge scenarios, examining the dynamic replication mechanism under dynamic client access patterns (Sec. V-D), and when supporting mobile clients which move across different locations and, consequently, nodes (Sec. V-E).

We compare `Arboreal` with other solutions that provide causal+ consistency on the edge, namely a decentralized solution using vector clocks (Engage [19]) and solutions using a centralized topology to enforce causality (Gesto [21] and Colony [20]). For the former we use the provided code, while

Authorized licensed use limited to: b-on: UNIVERSIDADE NOVA DE LISBOA. Downloaded on August 26,2024 at 17:51:24 UTC from IEEE Xplore. Restrictions apply.



(a) 20 nodes (b) 200 nodes

Fig. 5: Example node distributions in a geographic region

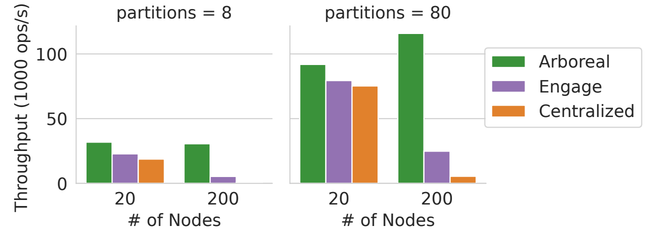


Fig. 6: Throughput of `Arboreal` versus causal+ solutions

for the latter, as the code is not available, we mimic their centralized topology by implementing a version of `Arboreal`, named *centralized*, where all edge locations connect directly to the data center. We also compare `Arboreal` against Cassandra [5], a cloud database that due to its configurable partial replication and peer-to-peer synchronization can be used in edge settings, having served as a baseline in contributions designed for both edge [29] and IoT [30].

A. Experimental Setup

We conducted experiments in a cluster of 10 machines, each having 2 AMD EPYC 7343 processors with 64 threads and 128 GB of memory, connected by a 20 Gbps network. We deployed a docker swarm across all machines, with an overlay network connecting all containers. A container with no resource restrictions on one machine models the data center, while up to 200 containers distributed across the others represent edge nodes. To better represent and edge environment, the computational resources of edge nodes were restricted to 2 virtual cores and 4 GB of memory. Each container executes an instance of `Arboreal`, as presented in Sec. IV.

To emulate a geographic region, we randomly distributed the 200 nodes across a virtual 2-dimensional space, representing edge locations, with the data center at the center. We used Linux *tc* to emulate latency between nodes based on their Euclidean distance, with a maximum latency of 150ms from an edge node to the data center. Experiments were conducted 3 times on 3 such distributions using either 20 or all 200 nodes. Fig. 5 shows an example distribution with the formed *control tree*, using the *deep* layout. Additionally, we deployed 200 client containers, distributing them across the same virtual space and setting the latency between each client and edge node using the same method, with a minimum latency of 10ms.

B. Performance

In our performance benchmarks, we evaluate the performance of `Arboreal` in terms of operation throughput and visibility times, comparing it against other causal+ solutions, Cassandra, and using different *control tree* layouts.

In these experiments, clients connect to their closest edge node and perform operations on data objects based on their location. The geographic region is divided into 8 equal segments, with each being assigned a data partition. Clients perform operations on data objects in their segment and the 2 adjacent segments. This setup assesses the performance of `Arboreal` with data locality, where clients are more likely to access nearby data objects. As the replication of data in `Arboreal` is dynamic and based on client access patterns, this results in each edge node replicating data objects from at least 3 partitions, with the data center replicating data objects from all 8 partitions. More complex scenarios with dynamic access patterns and client mobility are explored in Sec. V-D and V-E.

For Cassandra, as partial replication is based on a static placement, we configured each edge node as an independent cluster (*datacenter* in Cassandra terminology) and created partitions (*keyspaces* in Cassandra terminology) so that each edge node replicates all data objects from the 3 partitions accessed by clients, while the data center replicates all 8. A similar approach was used to distribute partitions in Engage.

1) Throughput: Fig. 6 shows the throughput of `Arboreal` compared to Engage and a centralized topology solution, varying the number of nodes and the number of distinct data partitions. We show the throughput of write operations only, as read operations execute locally in all solutions. Due to weak consistency allowing nodes to respond to clients without coordination with other nodes, measuring throughput on the clients is unreliable, as operations may be processed in their local nodes at a higher rate than they are replicated to other nodes. As such, the values displayed represent the maximum throughput measured in the data center node for each solution.

Two main conclusions can be drawn from these results: (1) Unlike existing causal+ solutions, `Arboreal` scales to hundreds of nodes without performance degradation. This is a result of avoiding both vector clocks (used in Engage) and centralized topologies (as the ones used in [21] and [20]) for causality enforcing, opting for a decentralized hierarchical topology. While it could be expected that a data center with high computing power could handle a centralized solution with a large number of edge nodes, this is not the case, as causality enforcement requires some form of (partial) serialization of operations, limiting parallelism; (2) Increasing the number of data partitions (reducing the number of data objects accessed by each client) and nodes (increasing the number of replicas for each data object) increases the throughput of `Arboreal`. This happens since each added node removes load from existing ones, allowing more operations to be processed in parallel. This is a key advantage of `Arboreal`'s dynamic

Authorized licensed use limited to: b-on: UNIVERSIDADE NOVA DE LISBOA. Downloaded on August 26,2024 at 17:51:24 UTC from IEEE Xplore. Restrictions apply.

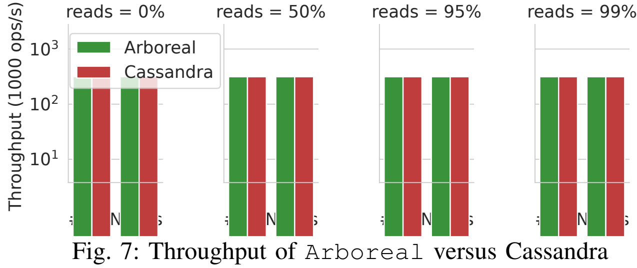


Fig. 7: Throughput of Arboreal versus Cassandra

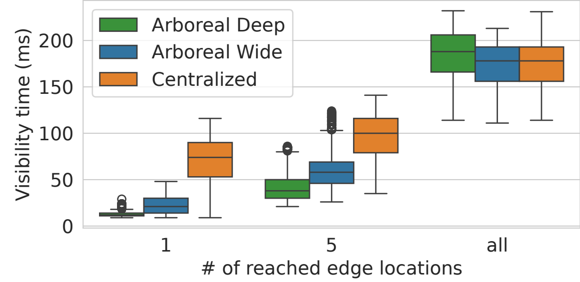


Fig. 8: Visibility times of different topologies

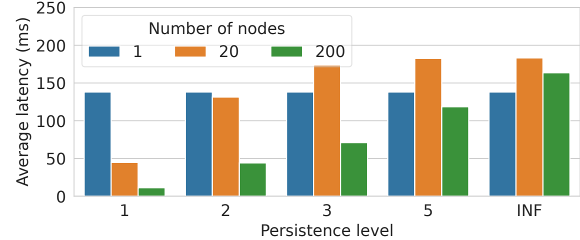


Fig. 9: Latency impact of persistence levels

replication mechanism over state-of-the-art solutions.

Fig. 7 shows the throughput achieved by Arboreal and Cassandra as observed by clients, varying the read/write ratio and number of nodes. Note the Y axis is in log scale. For more reliable measurements, we use ∞ persistence in Arboreal and *quorum* consistency in Cassandra, ensuring that writes are only acknowledged to clients after being replicated to the data center and a majority of edge nodes, respectively, preventing artificial throughput inflation. *Quorum* consistency in Cassandra also ensures clients observe the latest value of data objects, providing some consistency guarantees (although different from Arboreal, as it can violate causality across objects), making the comparison with Arboreal more fair.

In write-only scenarios (0% reads), Arboreal achieves higher throughput by leveraging its hierarchical topology and persistence mechanism. Nodes in Cassandra must send each write operation to all other nodes replicating the data object, then await acknowledgement from a quorum. In contrast, Arboreal sends write operations only to the parent (and any children replicating the object), and waits for the persistence acknowledgement. This greatly reduces message complexity, allowing higher throughput. With increased read operations, by processing reads locally, Arboreal outperforms Cassandra, which requires coordinating with a quorum before responding. While Cassandra can avoid coordination, sacrificing data consistency guarantees, to increase its throughput, Arboreal can process reads locally while providing causal+ consistency. Regardless Fig. 7 demonstrates that the hierarchical topology of Arboreal is more suitable for edge environments than traditional solutions designed for cloud environments.

2) Visibility Times and Tree Layouts: In this section, we explore the impact of different *control tree* layouts in Arboreal's hierarchical topology on the visibility times of operations and compare them with a centralized topology.

Utilizing the same setup as in the previous experiment, with 200 nodes and only write operations, Fig. 8 presents the results in the form of a boxplot, where each box shows the distribution of the visibility times of operations in the different topologies. *Arboreal deep* and *Arboreal wide* represent the layouts presented in Sec. IV, with the *wide* layout limited to a depth of 4. The figure depicts visibility times for the closest remote node (1), the 5th closest, and all nodes. The values for 1 and 5 are crucial in an edge environment as clients in geographical proximity, connected to different but close-by edge nodes, are likely to access the same data objects.

Results indicate that utilizing a hierarchical topology with the *deep* layout is optimal for achieving low visibility times, enabling rapid operation propagation to nearby nodes. However, reaching all nodes requires traversing a significant number of hops, resulting in higher global visibility times. In contrast, the centralized topology always requires propagating operations directly to the data center, resulting in much higher visibility times. The hierarchical *wide* layout serves as a balanced compromise, enabling slighty slower propagation to nearby nodes while matching the speed the *centralized* topology in reaching *all* nodes. Overall, these results show that a hierarchical topology is better suited for edge environments than a centralized one. In Secs. V-D and V-E, we further explore the benefits of Arboreal's hierarchical topology.

C. Fault Tolerance

In these experiments, we examine the resilience of Arboreal to the failure of edge locations, exploring different persistence levels, failure rates, and failure patterns. As reads are processed locally, all clients execute only write operations. The *deep* layout is used for the *control tree* in all experiments, allowing a wider range of persistence levels to be evaluated.

1) Operation Persistence: We start by evaluating the latency penalty of different persistence levels in Arboreal. Fig. 9 shows the latency of write operations when varying node numbers and persistence levels. In this experiment, persistence messages are propagated from nodes to their children every 20ms. The configurations with 1 node and those with a persistence level of 1 are used as baselines for comparison, representing the latency of operations if clients were communicating directly with the data center and without the persistence mechanism, respectively. As expected, as the persistence level increases, so does the latency of operations, requiring more

Authorized licensed use limited to: b-on: UNIVERSIDADE NOVA DE LISBOA. Downloaded on August 26,2024 at 17:51:24 UTC from IEEE Xplore. Restrictions apply.

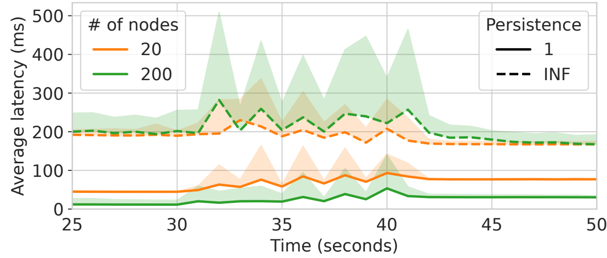
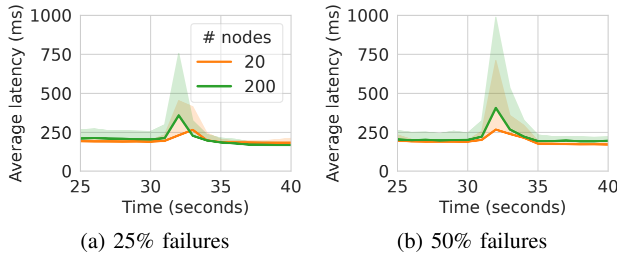


Fig. 10: Continuous node failures (50%)



(a) 25% failures

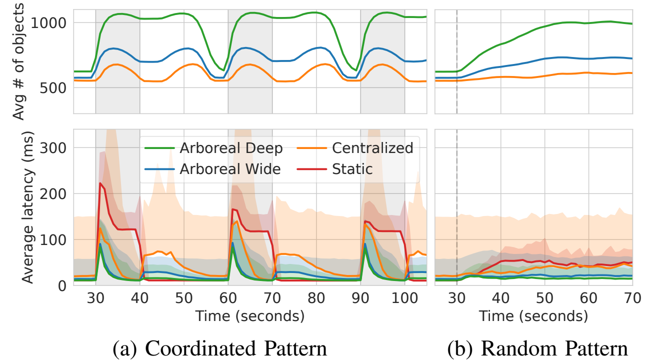
(b) 50% failures

Fig. 11: Simultaneous node failures with ∞ persistence

nodes to execute and acknowledge the operation before it is completed. In the 20 nodes experiments, as most nodes are within 3 hops from the data center, only persistence levels up to 2 provide latency benefits, with levels of 3 or above having similar latency to ∞. With 200 nodes, however, latency benefits are observed for persistence levels up to 5. This means that, even in very unstable edge environments where data persistence is crucial, using `Arboreal` can be beneficial when compared with directly contacting the data center, particularly in a real-world scenario where only a small fraction of data objects might need high levels of persistence.

2) Effects of failures: Fig. 10 shows the average latency over time as perceived by clients when 50% of all edge nodes fail within a 10-second period (30-40s in the figure), with persistence levels of 1 and ∞ and with both 20 and 200 nodes. The shaded area above each line represents the 99th percentile of latency (P99). In the results with persistence level of 1, the average latency barely increases during the failure event, as clients that are connected to non-failing nodes remain unaffected even if their node is temporarily disconnected from the *control tree*. However, after each node failure, some clients must reconnect to a new node. As outlined in Sec. III-C, the locality aspect of the *control tree* makes this process quick, with even the P99 not being significantly impacted. With a persistence level of ∞, the latency increase is more noticeable. This occurs because clients connected to nodes that did not fail, but whose ancestors did, will stop receiving replies to their operations until their branch of the *control tree* is repaired. As this process is fast and happens concurrently, the latency increase is minimal and does not accumulate over time.

Fig. 11 shows the effects of a large percentage of edge locations (25% and 50%) failing simultaneously, with clients using ∞ persistence level. The results highlight the crucial role



(a) Coordinated Pattern

(b) Random Pattern

Fig. 12: Dynamic client access patterns

of a decentralized solution for efficient failure recovery. In both experiments, average latency increases significantly during the failure events with larger failure rates causing higher latency penalties, as more clients are affected. However, the *control tree* recovery time remains mostly consistent, regardless of the initial number of nodes and the percentage of failures. Again, this happens since disconnected nodes can reconnect to the *control tree* in parallel and without coordination, enabling `Arboreal` to swiftly recover from failure events in highly unstable edge environments with numerous edge locations.

D. Dynamic Replication

While data locality is a primary assumption for edge computing (i.e., data is generated and consumed in the same geographical region), this does not mean that data is never accessed by far away clients. As an example, consider a social network application where users from a region might suddenly become interested in something that happened in another region. We explore how `Arboreal` adapts to such situations, analyzing the impact of different topologies on latency and object replicas, and comparing with solutions featuring static replication. Using the same data distribution as in Sec. V-B, where data partitions align with geographical segments, we modify client behavior to simulate the social network scenario. Clients predominantly access local data with a workload comprising 99% reads, occasionally expressing interest in non-local partitions. Fig. 12 reports the results of this experiment, using 200 nodes and a persistence level of 1, showing client-perceived latency over time (bottom plots) and the average number of replicated objects in all edge nodes. Two scenarios are considered: one where all clients synchronously change their interests to a new partition and later return to their original interests (12a), and another where clients randomly change their interests (12b). Similar to Sec. V-B2, we compare different layouts of `Arboreal`'s hierarchical topology against a centralized topology, all using dynamic replication, and against a traditional static replication solution (Cassandra).

In the coordinated scenario (Fig. 12a), shaded gray areas represent periods when clients access data from remote partitions. While latency always increases in these periods, in static

Authorized licensed use limited to: b-on: UNIVERSIDADE NOVA DE LISBOA. Downloaded on August 26,2024 at 17:51:24 UTC from IEEE Xplore. Restrictions apply.

replication solutions this increase is more pronounced and remains high until clients return to their original interests (i.e., access patterns). On the other hand, the dynamic replication mechanism of Arboreal results in smaller latency increase that quickly return to regular values. Considering different tree layouts, there is a trade-off between the number of replicas of data object and latency. While the *deep* layout results in lower latency, *wide* seems to provide the best trade-off, having a number of replicas close to the *centralized* and latency close to the *deep* layout.

In the random scenario (Fig. 12b), clients start accessing data from remote partitions at the 30 seconds mark. In Arboreal, the resulting latency increase is minimal, as the dynamic replication mechanism is reactively creating local replicas of data objects (as suggested by the increase in the P99). Using static replication, the average latency remains high as there are always clients accessing data from remote partitions. The *wide* layout again demonstrates an effective trade-off between latency and the number of object replicas.

These findings show that Arboreal is able to efficiently handle dynamic client access patterns, proving suitable for edge applications even when data locality is not guaranteed. Its hierarchical topology enables more efficient adaptation to such scenarios compared to solutions with centralized topologies.

E. Client Mobility

A key challenge for Arboreal is supporting mobile clients that change their connected edge location over time as they move. While crucial for applications such as location-based games (e.g., Pokémon Go) or autonomous vehicles, this is also relevant for any Internet service (e.g., social networks), as clients may move in their day-to-day lives. In this section, we evaluate how Arboreal's dynamic replication mechanism adapts to various client mobility patterns, comparing different layouts of its hierarchical topology with a centralized topology.

For this, we modeled clients that move around the geographical space, changing their connected edge node whenever a closer one is available, but never changing the data objects they are interested in. This forces Arboreal to continually create new replicas of the objects that clients are interested in and garbage-collect non-useful replicas. Fig. 13 reports the results of this experiment, showing the client-perceived latency across different mobility scenarios, using 200 nodes with a persistence level of 1, and a workload composed of 99% reads. Replicas of objects are garbage-collected after 5 seconds without being accessed. We modeled three mobility patterns: (1) a random pattern (Fig. 13a), where clients move randomly for 40 seconds, stop for 30 seconds, and repeat the process 2 more times; (2) a commute pattern (Fig. 13b), where clients move towards one of 5 fixed hot-spots (from second 30 to 60), and then return to their original position (from second 80 to 110); and (3) a mobile application pattern (Fig. 13c), where every 30 seconds, clients move together to a new hotspot in the geographical space, emulating location-based AR applications.

The results show that the dynamic replication mechanism of Arboreal is quick to adapt to new client positions, creating new replicas of data objects while garbage-collecting unneeded replicas. Due to the location-based topology of the *control tree*, when a client moves to a new edge node, both nodes likely share a close ancestor replicating the data objects of interest. This allows the new node to quickly create local replicas without being required to contact the cloud data center, avoiding high latency spikes, especially in the random and commute patterns. In the mobile application pattern, as all clients move together to the same small subset of edge nodes, the hierarchical topology advantages are less relevant, with the *centralized* topology showing slight lower latency values.

Whenever clients stop moving, latency quickly drops, showing that their accessed data objects have been replicated to their current edge location. This shows that Arboreal can adapt and converge in any mobility pattern, and its dynamic replication mechanism can efficiently handle mobile clients.

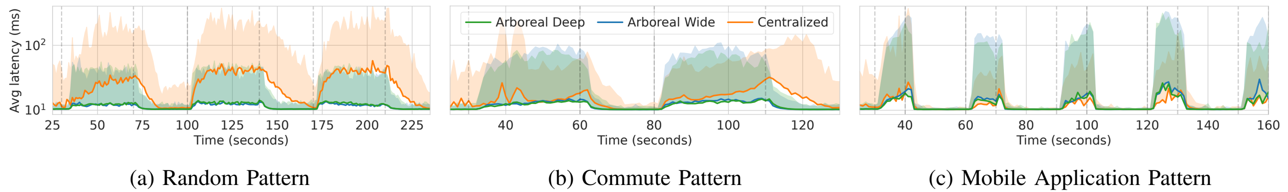
VI. Related Work

Causal+ Consistency: Various solutions for data replication with causal+ consistency have been proposed, with a majority of them focusing on data center deployments, limiting the number of replicas and lacking support for partial replication. Examples include COPS [12] and Eiger [31], which use explicit dependencies, or Orbe [17] and Cure [15], which rely on vector clocks that grow with the number of replicas or, in ChainReaction [32], with the number of data centers. Besides the lack of partial replication, the metadata overhead of these solutions is prohibitive for large-scale edge scenarios (as shown in V-B). Other solutions, like Saturn [18] and GentleRain [33], employ fixed-size metadata, limiting its overhead. However, they lack essential features to cope with edge deployments, such as reconfiguring after failures or partial replication, making them weak candidates for such scenarios.

Data Replication in the Edge: Various data management solutions with varying levels of data consistency have been proposed for the edge. Examples like DataFog [29] and CloudPath [34] lack any consistency guarantees, resulting in clients observing data anomalies. On the other hand, due to the highly geo-distributed nature of the edge, solutions providing strong consistency, such as FogStore [35], Colony [20], and DAST [36], only enforce their consistency guarantees within small, well-connected node groups, and not globally, as Arboreal does with causal+ consistency. Additionally, these solutions (like all strong consistency solutions) suffer from lack of availability during failures and lower fault-tolerance.

Regarding causal+ consistency on the edge, few solutions exist, with all having limitations to their applicability in the edge. Both Gesto [21] and Colony [20] rely on the datacenter to enforce causality guarantees, preventing direct cooperation between edge nodes. As shown in our evaluation, this impacts both performance and crucial edge aspects like client mobility. Engage [19] provides decentralized global causality in the edge, but relies on vector clocks which greatly limits its scalability. To the best of our knowledge, Arboreal is the first causal+ consistency solution able to efficiently handle deployments with hundreds of edge locations.

Authorized licensed use limited to: b-on: UNIVERSIDADE NOVA DE LISBOA. Downloaded on August 26,2024 at 17:51:24 UTC from IEEE Xplore. Restrictions apply.



(a) Random Pattern (b) Commute Pattern (c) Mobile Application Pattern

Fig. 13: Mobile clients with different mobility patterns

VII. Conclusion

This paper proposed `Arboreal`, a novel distributed data management system for enabling stateful edge applications. `Arboreal` addresses the key challenges for supporting this model, being the first solution to provide global causal+ consistency at scale while efficiently supporting client mobility due to its novel dynamic partial replication mechanism that adapts to their access patterns. This unique combination of features positions `Arboreal` as a suitable choice for diverse edge computing scenarios. Unlike existing causal+ solutions, `Arboreal` takes full advantage of the edge computing paradigm, by operating fully decentralized, eliminating the need for cloud mediation. Experimental results show `Arboreal`'s superior performance in various scenarios, handling mobile clients with ease, efficiently dealing with abrupt changes on client access patterns, and swiftly reconfiguring after failures with minimal impact on clients.

References

[1] Y. Siriwardhana, P. Porambage, M. Liyanage, and M. Ylianttila, "A survey on mobile augmented reality with 5g mobile edge computing," *IEEE Communications Surveys & Tutorials*, vol. 23, no. 2, 2021.

[2] S. Liu *et al.*, "Edge computing for autonomous driving: Opportunities and challenges," *Proceedings of the IEEE*, vol. 107, no. 8, 2019.

[3] R. Bhardwaj *et al.*, "Ekya: Continuous learning of video analytics models on edge compute servers," in *19th USENIX Symposium on Networked Systems Design and Implementation*, 2022.

[4] G. DeCandia *et al.*, "Dynamo: Amazon's highly available key-value store," *SIGOPS Oper. Syst. Rev.*, vol. 41, no. 6, oct 2007.

[5] A. Lakshman and P. Malik, "Cassandra: a decentralized structured storage system," *ACM SIGOPS Oper. Syst. Rev.*, vol. 44, no. 2, 2010.

[6] J. C. Corbett *et al.*, "Spanner: Google's globally distributed database," *ACM Transactions on Computer Systems*, vol. 31, no. 3, 2013.

[7] B. Schlinker *et al.*, "Engineering egress with edge fabric: Steering oceans of content to the world," in *Proceedings of the Conference of the ACM Special Interest Group on Data Communication*, 2017.

[8] K.-K. Yap *et al.*, "Taking the edge off with espresso: Scale, reliability and programmability for global internet peering," in *Conference of the ACM Special Interest Group on Data Communication*, 2017.

[9] "Lambda@edge documentation," https://aws.amazon.com/lambda/edge/.

[10] "Google peering documentation," https://peering.google.com.

[11] D. Chou *et al.*, "Taiji: managing global user traffic for large-scale internet services at the edge," in *27th ACM Symposium on Operating Systems Principles*, 2019.

[12] W. Lloyd, M. Freedman, M. Kaminsky, and D. Andersen, "Don't settle for eventual: Scalable causal consistency for wide-area storage with cops," in *23rd ACM Symposium on Operating Systems Principles*, 2011.

[13] L. Lamport, "Time, clocks, and the ordering of events in a distributed system." Association for Computing Machinery, jul 1978, vol. 21, no. 7.

[14] H. Attiya, F. Ellen, and A. Morrison, "Limitations of highly-available eventually-consistent data stores," in *Proceedings of the 2015 ACM Symposium on Principles of Distributed Computing*, 2015.

[15] D. D. Akkoorath *et al.*, "Cure: Strong semantics meets high availability and low latency," in *2016 IEEE 36th International Conference on Distributed Computing Systems*. IEEE, 2016.

[16] S. A. Mehdi *et al.*, "I cant believe its not causal! scalable causal consistency with no slowdown cascades," in *14th USENIX Symposium on Networked Systems Design and Implementation*, 2017.

[17] J. Du, S. Elnikety, A. Roy, and W. Zwaenepoel, "Orbe: Scalable causal consistency using dependency matrices and physical clocks," in *4th annual Symposium on Cloud Computing*, 2013.

[18] M. Bravo, L. Rodrigues, and P. Van Roy, "Saturn: A distributed metadata service for causal consistency," in *Proceedings of the Twelfth European Conference on Computer Systems*, 2017.

[19] M. Belém *et al.*, "Engage: Session guarantees for the edge," in *2022 Inter. Conf. on Computer Communications and Networks*. IEEE, 2022.

[20] I. Toumlilt, P. Sutra, and M. Shapiro, "Highly-available and consistent group collaboration at the edge with colony," in *Proceedings of the 22nd International Middleware Conference*, 2021.

[21] N. Afonso, M. Bravo, and L. Rodrigues, "Combining high throughput and low migration latency for consistent data storage on the edge," in *29th IEEE Int. Conf. on Computer Communications and Networks*, 2020.

[22] Z. Jia and E. Witchel, "Boki: Stateful serverless computing with shared logs," in *ACM SIGOPS 28th Symposium on Oper. Syst. Principles*, 2021.

[23] A. van der Linde, P. Fouto, J. Leitão, and N. Preguiça, "The intrinsic cost of causal consistency," in *Proceedings of the 7th Workshop on Principles and Practice of Consistency for Distributed Data*, 2020.

[24] S. S. Kulkarni *et al.*, "Logical physical clocks," in *Principles of Distributed Systems: 18th International Conference*. Springer, 2014.

[25] J. Leitao, P. Costa, M. Gomes, and N. Preguiça, "Towards enabling novel edge-enabled applications," *arXiv preprint arXiv:1805.06989*, 2018.

[26] "Netty project," https://netty.io/.

[27] J. Leitao, J. Pereira, and L. Rodrigues, "Hyparview: A membership protocol for reliable gossip-based broadcast," in *37th International Conference on Dependable Systems and Networks*, 2007.

[28] B. F. Cooper, A. Silberstein, E. Tam, R. Ramakrishnan, and R. Sears, "Benchmarking cloud serving systems with ycsb," in *Proceedings of the 1st ACM symposium on Cloud computing*, 2010.

[29] H. Gupta, Z. Xu, and U. Ramachandran, "Datafog: Towards a holistic data management platform for the iot age at the network edge," in *USENIX Workshop on Hot Topics in Edge Computing*, 2018.

[30] L. Silva and J. Lima, "An evaluation of cassandra nosql database on a low-power cluster," in *Symposium on Computer Architecture and High Performance Computing Workshops*, 2021.

[31] W. Lloyd, M. J. Freedman, M. Kaminsky, and D. G. Andersen, "Stronger semantics for low-latency geo-replicated storage," in *10th USENIX Symposium on Networked Systems Design and Implementation*, 2013.

[32] S. Almeida, J. Leitão, and L. Rodrigues, "Chainreaction: a causal+ consistent datastore based on chain replication," in *Proceedings of the 8th ACM European Conference on Computer Systems*, 2013.

[33] J. Du, C. Iorgulescu, A. Roy, and W. Zwaenepoel, "Gentlerain: Cheap and scalable causal consistency with physical clocks," in *Proceedings of the ACM Symposium on Cloud Computing*, 2014.

[34] S. H. Mortazavi, M. Salehe, C. S. Gomes, C. Phillips, and E. De Lara, "Cloudpath: A multi-tier cloud computing framework," in *Second ACM/IEEE Symposium on Edge Computing*, 2017.

[35] H. Gupta and U. Ramachandran, "Fogstore: A geo-distributed key-value store guaranteeing low latency for strongly consistent access," in *12th ACM International Conf. on Distributed and Event-based Systems*, 2018.

[36] X. Chen *et al.*, "Achieving low tail-latency and high scalability for serializable transactions in edge computing," in *Sixteenth European Conference on Computer Systems*, 2021.

Authorized licensed use limited to: b-on: UNIVERSIDADE NOVA DE LISBOA. Downloaded on August 26,2024 at 17:51:24 UTC from IEEE Xplore. Restrictions apply.