Lattice: A Decentralized, Distributed Datastore

Computes, inc.

February 2018 [*]

Abstract

Lattice is a decentralized, distributed datastore that is used to build a distributed work queue, ledger, and general-purpose datastore for the Computes mesh computer[?] . The Lattice protocol allows authorized computers to self-organize into a mesh computer, limited only by the number and power of the members. Lattice will intelligently allocate work to the best members of the mesh, based on the requirements of the task.

Lattice is a layer on top of IPFS[?] , which provides a peer-to-peer distributed filesystem, including a Merkle DAG[?] . Utilizing IPFS as a subsystem allows Lattice to greatly benefit from the novel and essential features provided. Embracing the content addressed, immutable nature of IPFS also allows Lattice to ensure data integrity and accuracy. Lattice introduces the novel concepts: Grow Only Hash Object and the Hash Pointer Collaboration Protocol, Distributed Trust System.

The Grow Only Hash Object (GOHO) is a conflict-free data structure[?] composed of nodes in the IPFS Merkle DAG that can be resolved into a map-like structure. Members of the Computes mesh can add data to a GOHO without resolving the entire dataset, eliminating inefficient use of computation and bandwidth.

The Hash Pointer Collaboration Protocol (HPCP) allows member nodes to query and update other members about the latest root hash of their Merkle DAG. This allows efficient and fast synchronization of Merkle DAGs across peers.

The Distributed Trust System (DTS) allows members to earn and distribute trust by verifying work done by other members.

[*]Lattice is a work in progress. New versions of this paper will appear at `https://www.computes.com/whitepapers`. For comments or corrections, contact us at `whitepaper@computes.com`.

Contents

List of Figures

1 Introduction

There are numerous implementations of distributed work queues, nearly all of them require a centralized queue manager to assign and distribute work. Lattice allows members of the Computes mesh to collaborate peer-to-peer to assemble the queue, enqueue, dequeue, and complete work. Lattice is highly resilient and allows members to leave and join at will.

1.1 Components

Lattice is constructed from base components inherited from IPFS along with three novel components.

1. **??**: Utilizing the IPFS DAG and CRDT techniques, members can easily modify a shared, distributed map-like object without conflict and without accessing the entire object.
2. **??**: Utilizing IPFS Pub/Sub members can query and share the latest Merkle Root of the Grow Only Hash Object with other members.
3. **??**: When a new Merkle Root is detected for a Grow Only Hash Object, a set of conditions are evaluated, and new work may be enqueued into Lattice.

2 Grow Only Hash Object

A Grow Only Hash Object (GOHO) is a conflict-free replicated data type (CRDT) assembled from leaf nodes of a Merkle DAG. The GOHO allows clients to determine if data duplication is desired or not by setting a unique `idempotentKey`.

2.1 Grow Only Hash Object Protocol

(Append, Get, Set)

- Append(key, data, idempotentKey): Clients execute the Append protocol to *add* `data` to a set. The key is a path-like structure that allows for setting subkeys of the map-like data structure. The `idempotentKey` allows the client to prevent data duplication by using the same key, data, `idempotentKey` triple.

- Get() $\rightarrow$ data: Clients execute the Get protocol to *resolve* the Merkle DAG into a map-like data structure.

- Set(key, data, idempotentKey): Clients execute the Set protocol to *set* a `key` to the `data`. The `key` is a path-like structure that allows for setting subkeys of the map-like data structure. The `idempotentKey` allows the client to prevent data duplication by using the same `key`, `data`, `idempotentKey` triple.

2.2 Simple Example

Figure **??** shows a Grow Only Hash Object that has a new node appended to it. For a more complex example, please see **??**.

2.3 Merge Example

Figure **??** shows two clients C_1 and C_2 updating the same GOHO and merging the changes into a new tree.

3 Hash Pointer Collaboration Protocol

The Hash Pointer Collaboration Protocol (HPCP) allows clients to track the latest Merkle Root without synchronization across the Computes mesh. Knowledge of the Origin hash is required to track changes. Clients will establish a shared communication channel to share updates and queries. Clients need not know the most recent merkle root and can trust other peers to merge the update with the published update.

3.1 Hash Pointer Collaboration Protocol

(Query, Update)

- Query(genesis): Clients execute the Query protocol to *retrieve* the latest merkle root from other peers.
- Update(genesis, hash): Clients execute the Update protocol to *notify* peers of a new root for a given origin.

3.2 Conflicts

When a Client receives a previously unknown hash from Update, it will merge the Merkle DAG into its own copy. If the union of the two trees results in a unique dataset, the Client will publish an Update with the new hash.

4 Distributed Trust System

The Distributed Trust System Protocol (DTS) allows members of the Computes Mesh to earn and distribute trust by verifying work done by other members of the mesh.

4.1 Distributed Trust System Protocol

(Trust, Verify)

- Trust(memberId): Clients execute the Trust protocol to *give* trust to another member of the Computes Mesh.
- Verify(memberId, task): Clients execute the Verify protocol to *execute* a given task again and checking that the results match the previous run by a given member.

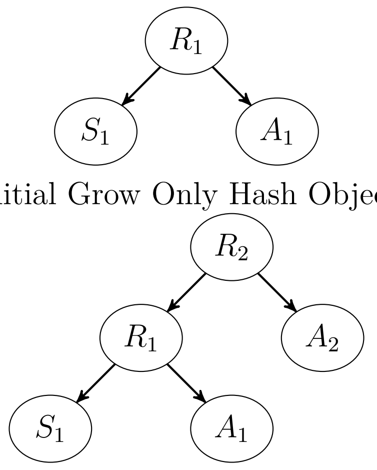

Figure 1: Simple Grow Only Hash Object Example

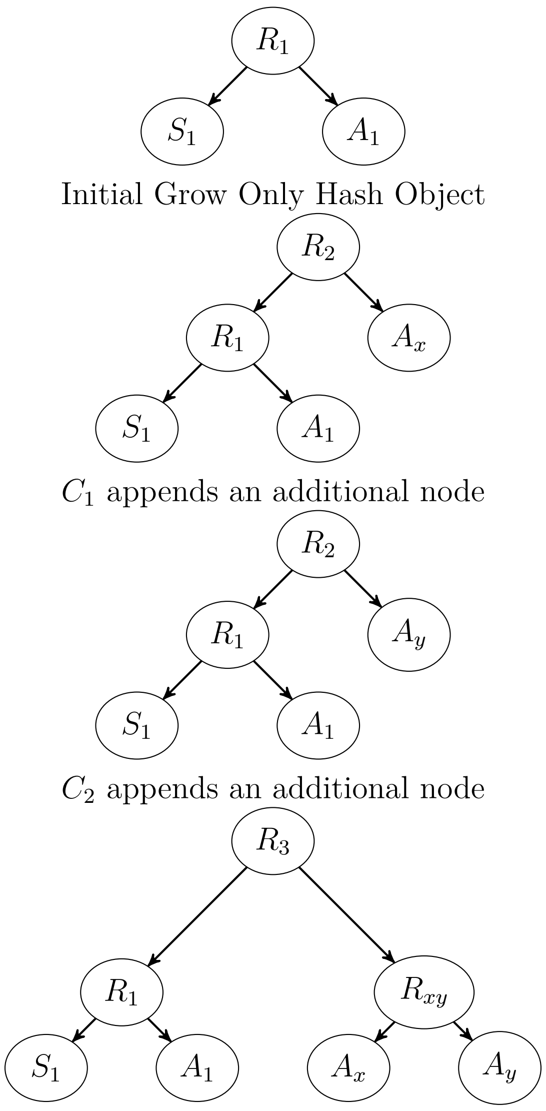

Figure 2: Grow Only Hash Object Merge Example