Anti-entropy Through Merkle Trees

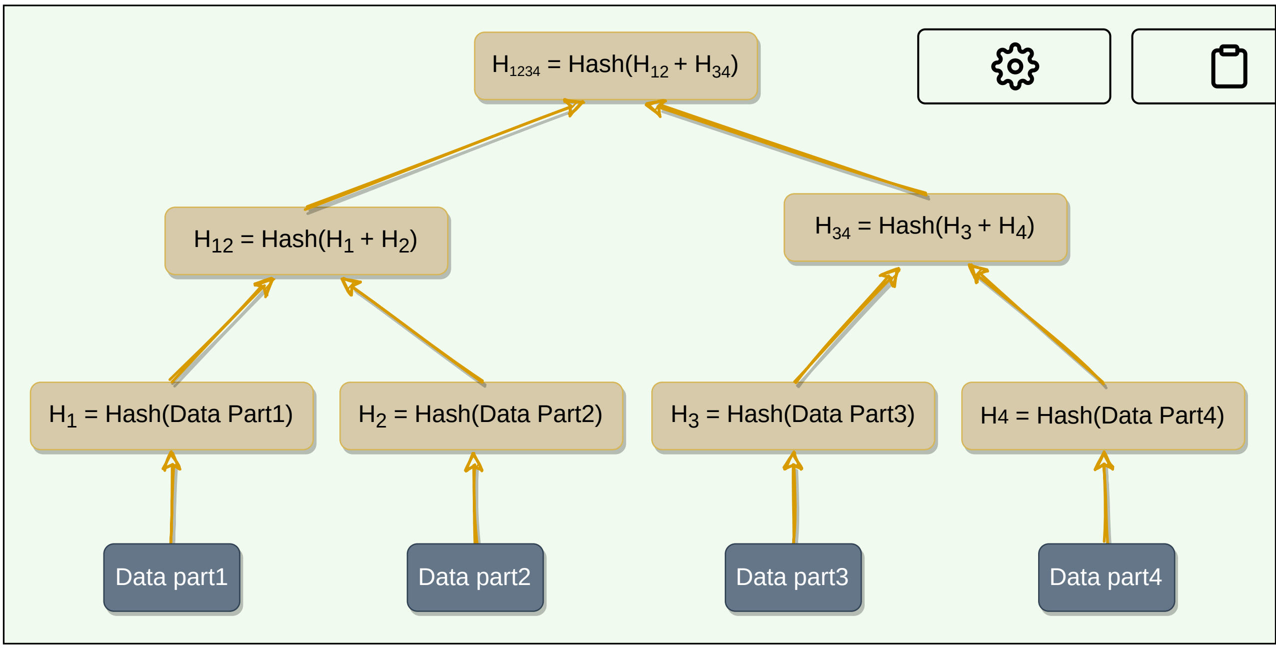
Let's understand how Dynamo uses Merkle trees for anti-entropy operations.

We'll cover the following

As we know, Dynamo uses vector clocks to remove conflicts while serving read requests. Now, if a replica falls significantly behind others, it might take a very long time to resolve conflicts using just vector clocks. It would be nice to be able to automatically resolve some conflicts in the background. To do this, we need to quickly compare two copies of a range of data residing on different replicas and figure out exactly which parts are different.

What are Merkle trees?#

A replica can contain a lot of data. Naively splitting up the entire data range for checksums is not very feasible; there is simply too much data to be transferred. Therefore, Dynamo uses **Merkle trees** to compare replicas of a range. A Merkle tree is a binary tree of hashes, where each internal node is the hash of its two children, and each leaf node is a hash of a portion of the original data.



Merkle tree

Comparing Merkle trees is conceptually simple:

1. Compare the root hashes of both trees.
2. If they are equal, stop.
3. Recurse on the left and right children.

Ultimately, this means that replicas know precisely which parts of the range are different, and the amount of data exchanged is minimized.

Merits and demerits of Merkle trees#

The principal advantage of using a Merkle tree is that each branch of the tree can be checked independently without requiring nodes to download the entire tree or the whole data set. Hence, Merkle trees minimize the amount of data that needs to be transferred for synchronization and reduce the number of disk reads performed during the anti-entropy process.

The disadvantage of using Merkle trees is that many key ranges can change when a node joins or leaves, and as a result, the trees need to be recalculated.

← Back

The Life of Dynamo’s put() & get() Op...

Next →

Gossip Protocol

Mark as Completed

Report an Issue