Merkle-tree-based integrity verification protocol for geo-distributed storage systems

Santo Cariotti

Alma Mater Studiorum
Università di Bologna

October 30, 2025

Content

1. What kind of problem are we facing?

Content

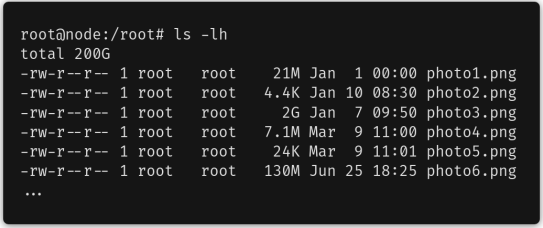
1. What kind of problem are we facing?
2. How did we manage to solve this?

Content

1. What kind of problem are we facing?
2. How did we manage to solve this?
3. How about scalability?

Problem

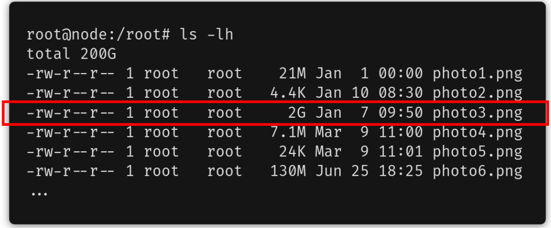
Given a filesystem, how can I be sure that every file keeps integrity?



```
root@node:/root# ls -lh
total 200G
-rw-r--r-- 1 root   root    21M Jan  1 00:00 photo1.png
-rw-r--r-- 1 root   root   4.4K Jan 10 08:30 photo2.png
-rw-r--r-- 1 root   root     2G Jan  7 09:50 photo3.png
-rw-r--r-- 1 root   root   7.1M Mar  9 11:00 photo4.png
-rw-r--r-- 1 root   root    24K Mar  9 11:01 photo5.png
-rw-r--r-- 1 root   root   130M Jun 25 18:25 photo6.png
...
```

Problem

Given a filesystem, how can I be sure that every file keeps integrity?

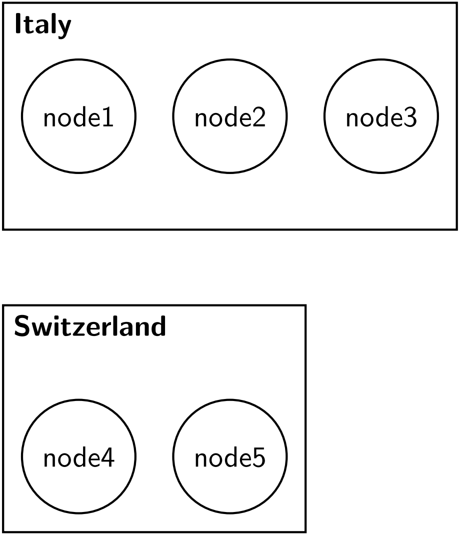


```
root@node:/root# ls -lh
total 200G
-rw-r--r-- 1 root   root    21M Jan  1 00:00 photo1.png
-rw-r--r-- 1 root   root   4.4K Jan 10 08:30 photo2.png
-rw-r--r-- 1 root   root     2G Jan  7 09:50 photo3.png
-rw-r--r-- 1 root   root   7.1M Mar  9 11:00 photo4.png
-rw-r--r-- 1 root   root    24K Mar  9 11:01 photo5.png
-rw-r--r-- 1 root   root   130M Jun 25 18:25 photo6.png
 ...
```

We could use a checksum system to find out that the bordered file is corrupted.

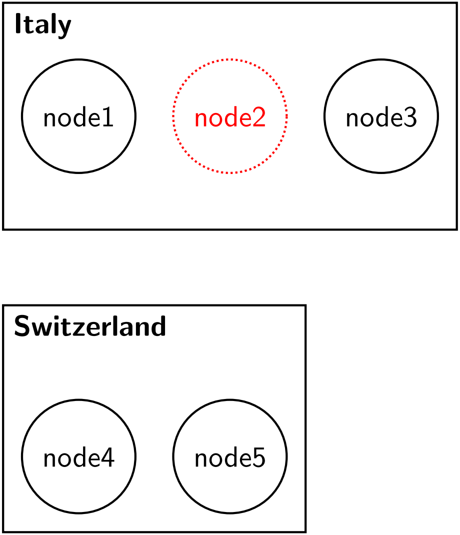
Problem

But, if we have more than one filesystem distributed across different regions?



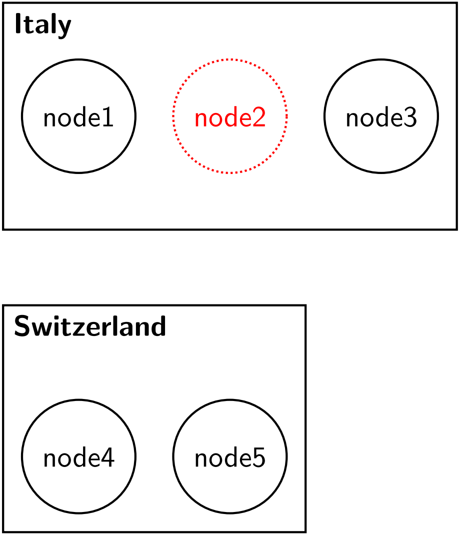
Problem

But, if we have more than one filesystem distributed across different regions? We should make a checksum check for each node.



Problem

But, if we have more than one filesystem distributed across different regions? We should make a checksum check for each node. **Too complex.**



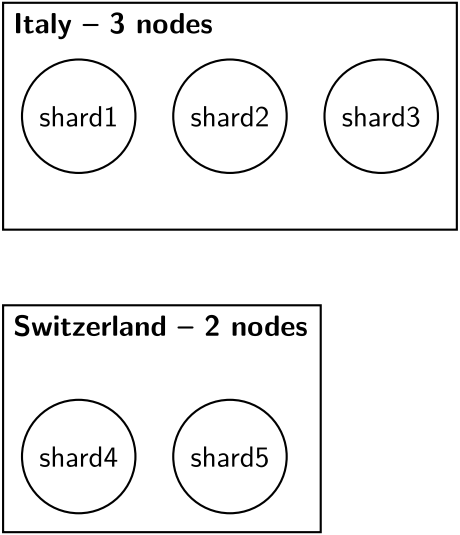
Cubbit

Cubbit, a geo-distributed storage system.

Cubbit – How it works

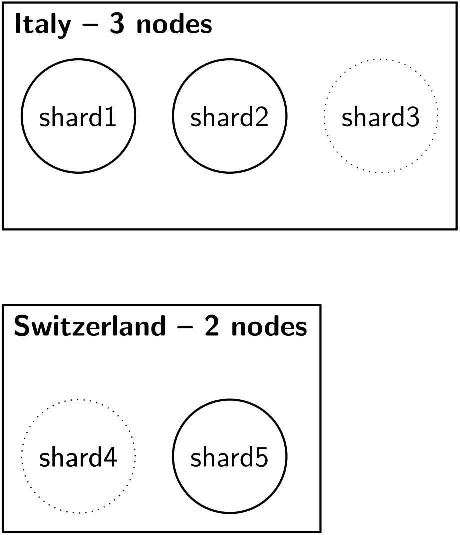
A file is split in $n + k$ shards using Reed-Solomon codes and each shard is sent to a different node.

We need at least n shards to reconstruct the entire file, k shards are used as redundancy.



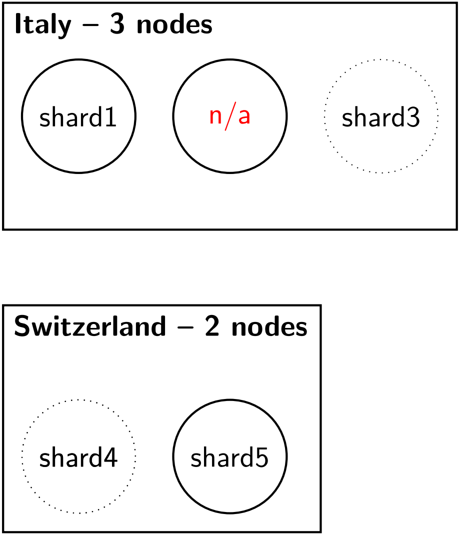
Cubbit – Problems using checksum

1. If nodes are offline, can't check all shards for a file.



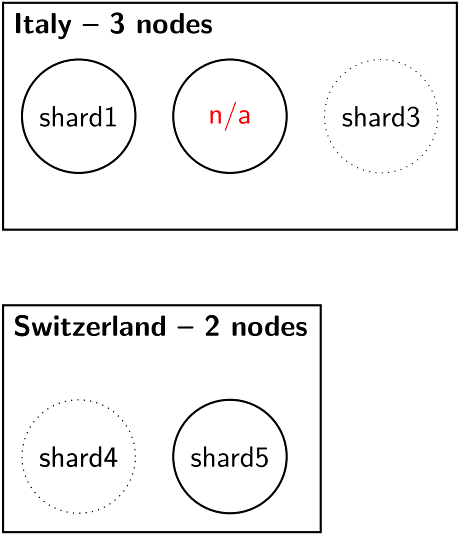
Cubbit – Problems using checksum

1. If nodes are offline, can't check all shards for a file.
2. During an upload some agents can be offline, but they could be online during the check.



Cubbit – Problems using checksum

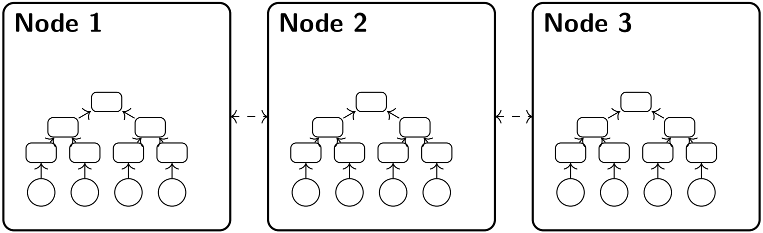
1. If nodes are offline, can't check all shards for a file.
2. During an upload some agents can be offline, but they could be online during the check.
3. Check for each reconstructed file or for each shard?



Solution

Solution

Each node uses a Merkle-tree-structure to organize shards during the integrity verification. Every node agree on what file is corrupted thanks to Raft. Data are organized using Reed-Solomon codes.

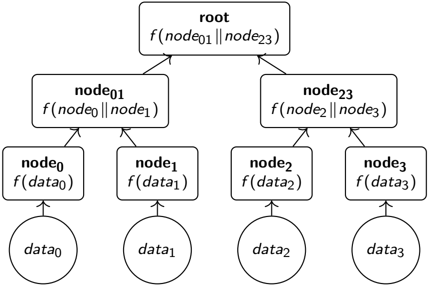


Some background

Merkle trees

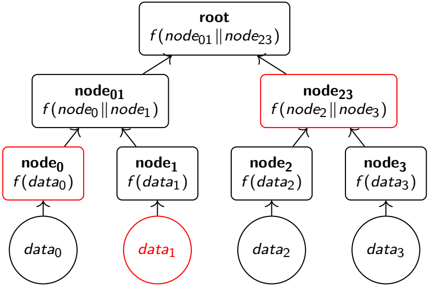
It is a binary tree T of height H with 2^H leaves and $2^H - 1$ internal nodes. Each leaf stores the cryptographic hash of the underlying data, rather than the raw data itself. The same cryptographic hash function is applied recursively at internal nodes, which store the hash of the concatenation of their two children.

$$n_{parent} = f(n_{left} || n_{right})$$



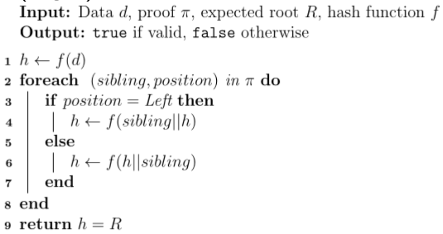
Merkle trees – Proof

Merkle trees has the ability to prove that a given piece of data is part of a larger set, without revealing or recomputing the entire dataset. For $data_1$ we have $\pi = \{node_0, node_{23}\}$.



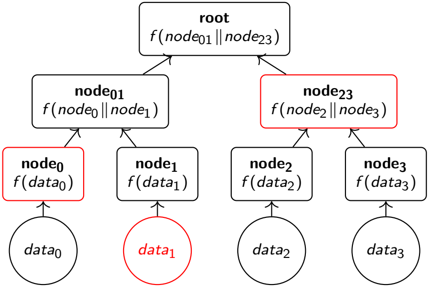
Merkle trees – Proof

Given a path π we can make a proof verification comparing the result with the presumed root in $O(\log n)$.



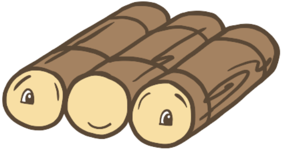
Input: Data d, proof π, expected root R, hash function f
Output: `true` if valid, `false` otherwise

```
h ← f(d)
foreach (sibling, position) in π do
    if position = Left then
        h ← f(sibling||h)
    else
        h ← f(h||sibling)
    end
end
return h = R
```



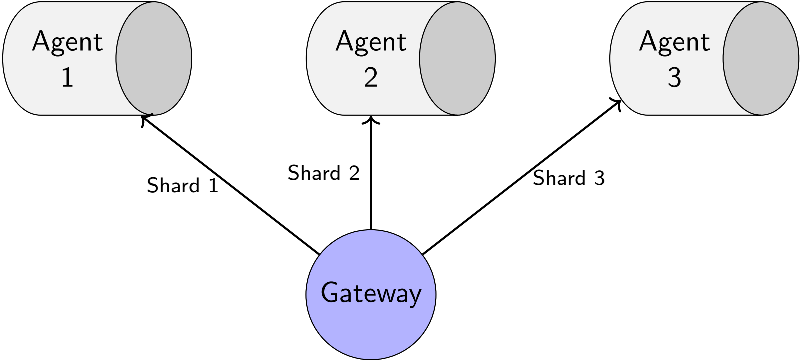
Raft

A consensus protocol, where each server on a cluster is a follower, a candidate or a leader. There is only leader and it is responsible to send messages to other servers via a log.



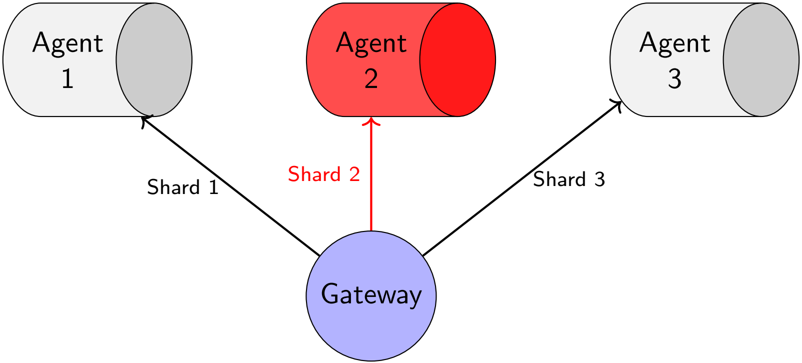
Reed-Solomon

During an upload, a file is split in $n + k$ shards and send each shard to a different node.



Reed-Solomon

Up to k agents could be offline, and the upload/download of files still works. We should make a recoverage of the missing shard when the agent comes back online.


Agent 1
Agent 2
Agent 3
Shard 1
Shard 2
Shard 3
Gateway