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

Presented by Santo Cariotti
Supervised by Prof. Özalp Babaoğlu

Alma Mater Studiorum
Università di Bologna

October 30, 2025

Problem

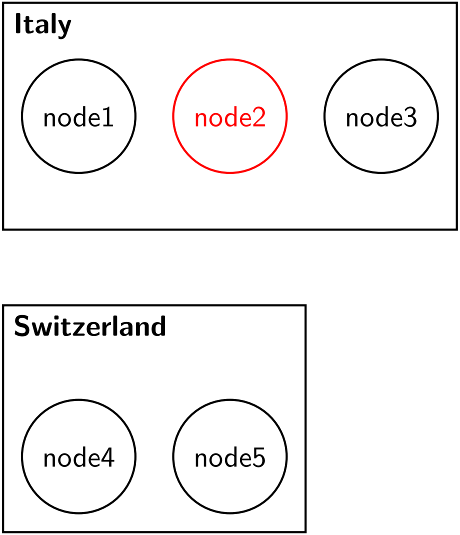
How can we verify that none of the files are corrupted?

```
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 can use checksums.

Problem

But, what if we have a file split in pieces and distributed across different regions? We would need to perform a checksum on each node.



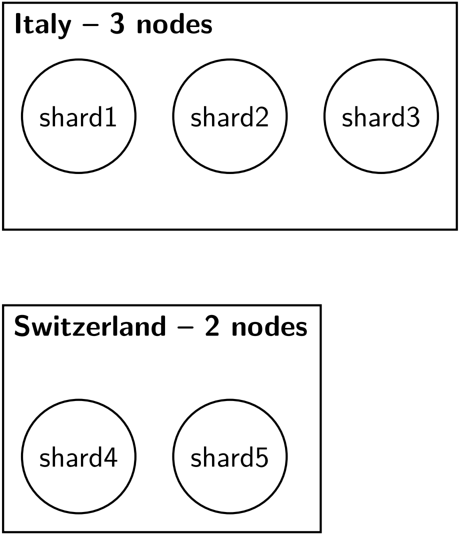
Cubbit

Cubbit

Cubbit, a geo-distributed storage system.

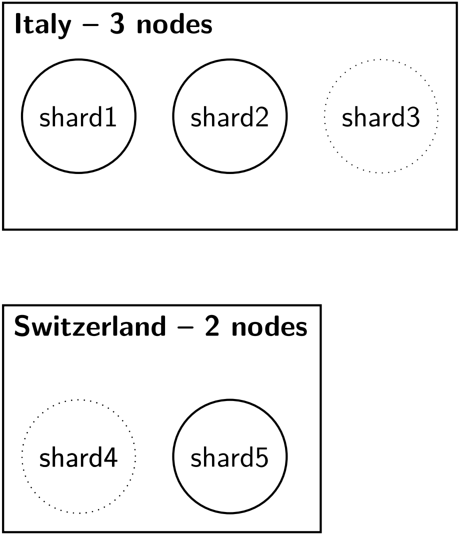
Cubbit – How it works

Each file is split in $n + k$ shards (i.e., pieces) using Reed-Solomon. Each shard is sent to a different node (i.e., agent).
At least n shards are required to reconstruct the file, while k shards provide redundancy.



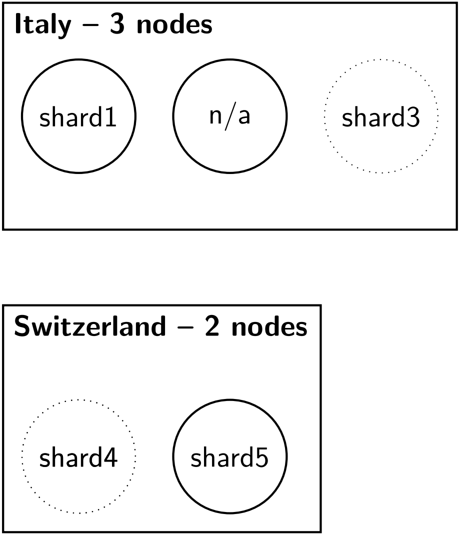
Cubbit – Problems using checksum

1. If nodes are offline, it becomes impossible to check all shards for a file.



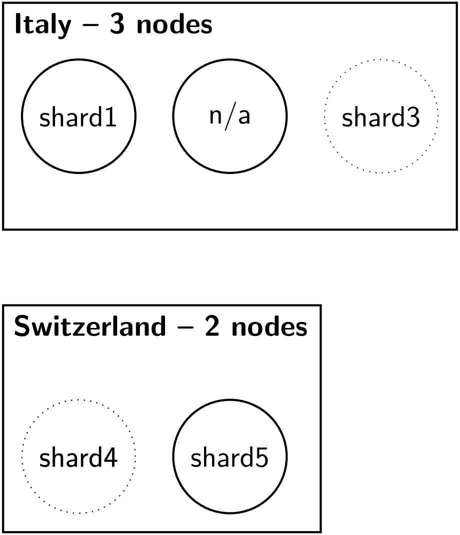
Cubbit – Problems using checksum

1. If nodes are offline, it becomes impossible to check all shards for a file.
2. During an upload some nodes can be offline, but they could be online during the check.



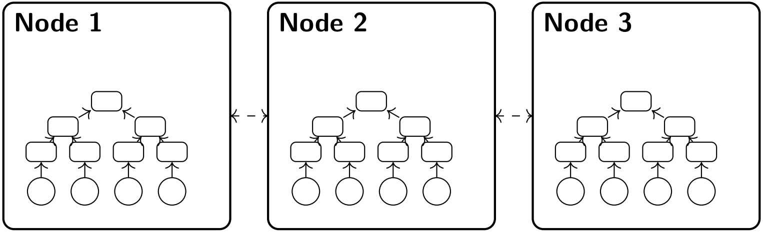
Cubbit – Problems using checksum

1. If nodes are offline, it becomes impossible to check all shards for a file.
2. During an upload some nodes can be offline, but they could be online during the check.
3. Check for each reconstructed file or for each shard?



Solution

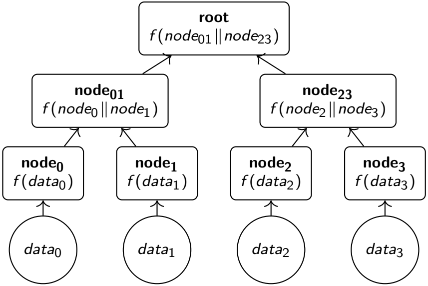
Each node uses a Merkle tree structure to organize shards during integrity verification. All nodes agree on which file is corrupted through Raft consensus. Data are organized using Reed-Solomon codes.



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.

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



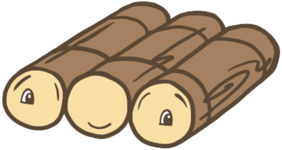
Merkle trees – Lib

A dedicated Rust library was developed to efficiently compute Merkle tree root hashes.

Hash function	5 MB	10 MB	15 MB
SHA-256	89.901 ms	178.42 ms	268.53 ms
Keccak-256	521.49 ms	1.1334 s	1.3438 s
BLAKE3	73.091 ms	154.68 ms	219.79 ms

Raft

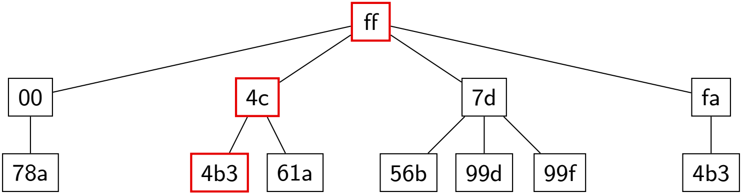
A consensus protocol, where each server on a cluster is a follower, a candidate or a leader. There is a single leader, responsible for sending messages to other servers via the Raft log.



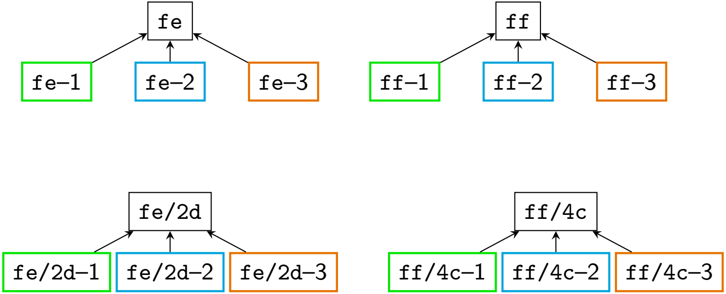
Upload flow

Let's save our Christmas holidays family picture on Cubbit.

- Select a file. (e.g., xmas2024.png)
- Gateway applies Reed-Solomon codes and sends each shard to a different agent. (e.g., xmas2024.1 to Agent 1)
- Each agent chooses the same pseudo-random string for the filename. (e.g., ff4c4b3)
- Finally, every agent saves the file using a two-level folder structure. (e.g. ff/4c/4b3.1 for Agent 1)



Aggregated roots

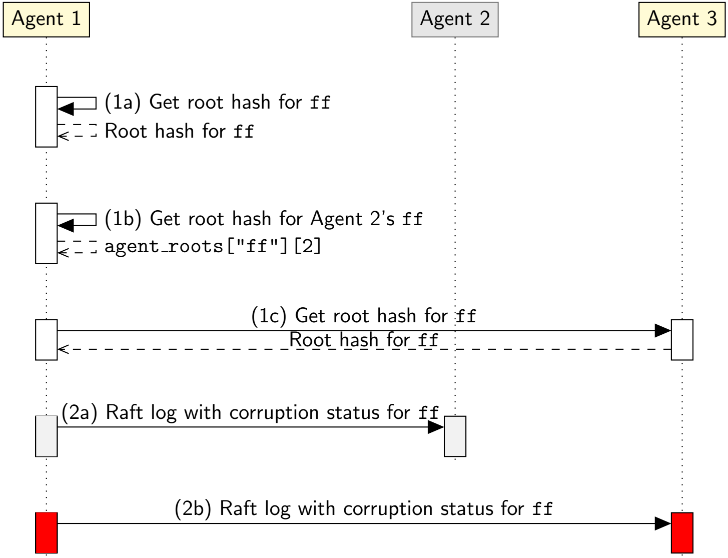


The leader builds Merkle trees for each folder, with the agents' Merkle root hashes as leaves.

Map of hashes

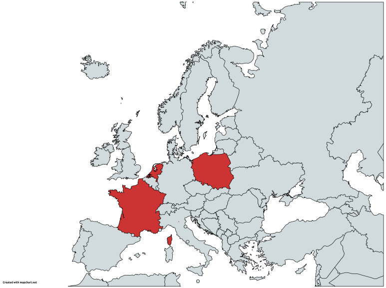
```
roots = {
    "fe":   "root hash of fe",
    "ff":   "root hash of ff",
    "ff/4c":"root hash of ff/4c",
    # ...
}

agent_roots = {
    "fe": [
        "Agent 1 root hash of fe",
        "Agent 2 root hash of fe",
        "Agent 3 root hash of fe"
    ],
    # ...
    "ff/4c": [
        "Agent 1 root hash of ff/4c",
        "Agent 2 root hash of ff/4c",
        "Agent 3 root hash of ff/4c"
    ],
}
```

Check corruptions



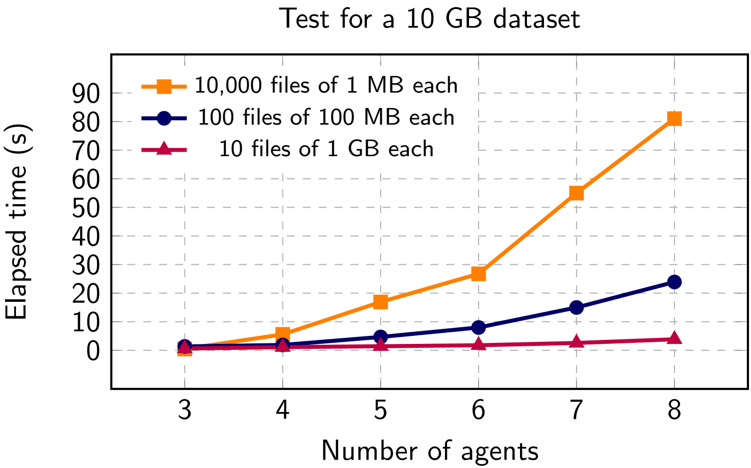
Tests – Set up

8 Agents with 4 CPU(s), 8-16 GB of RAM, and 45 GB of Disk.

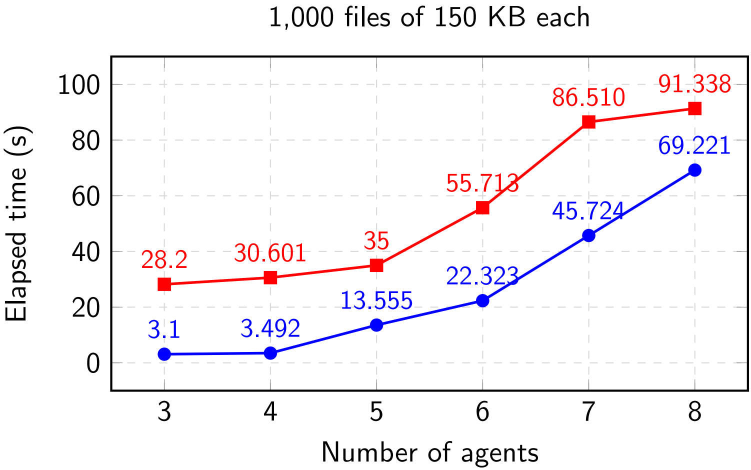


For each test, we consider the number of online/offline agents and the file sizes and counts.

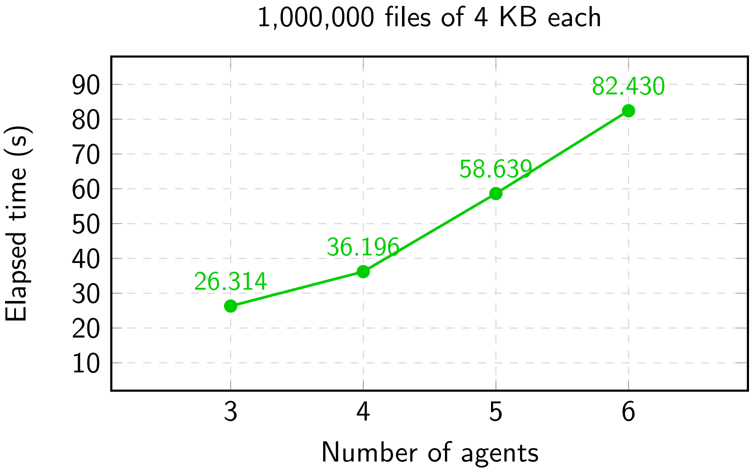
Same dataset but different file size and count



Test with offline agents



Test with very large number of tiny files



Results

- Scenarios with many small files amplify the cost of synchronization and consensus.
- Verification time increases with the number of agents and files.
- Confirmed the correctness and resilience of the approach under different scenarios and network conditions.

Conclusion

By combining Merkle trees, Raft consensus, and Reed-Solomon codes, we built a scalable and fault-tolerant protocol for distributed data integrity verification without relying on full file scans or constant node availability.

Thank you!