Design and implementation of ElmerFS

...

Romain Vaillant

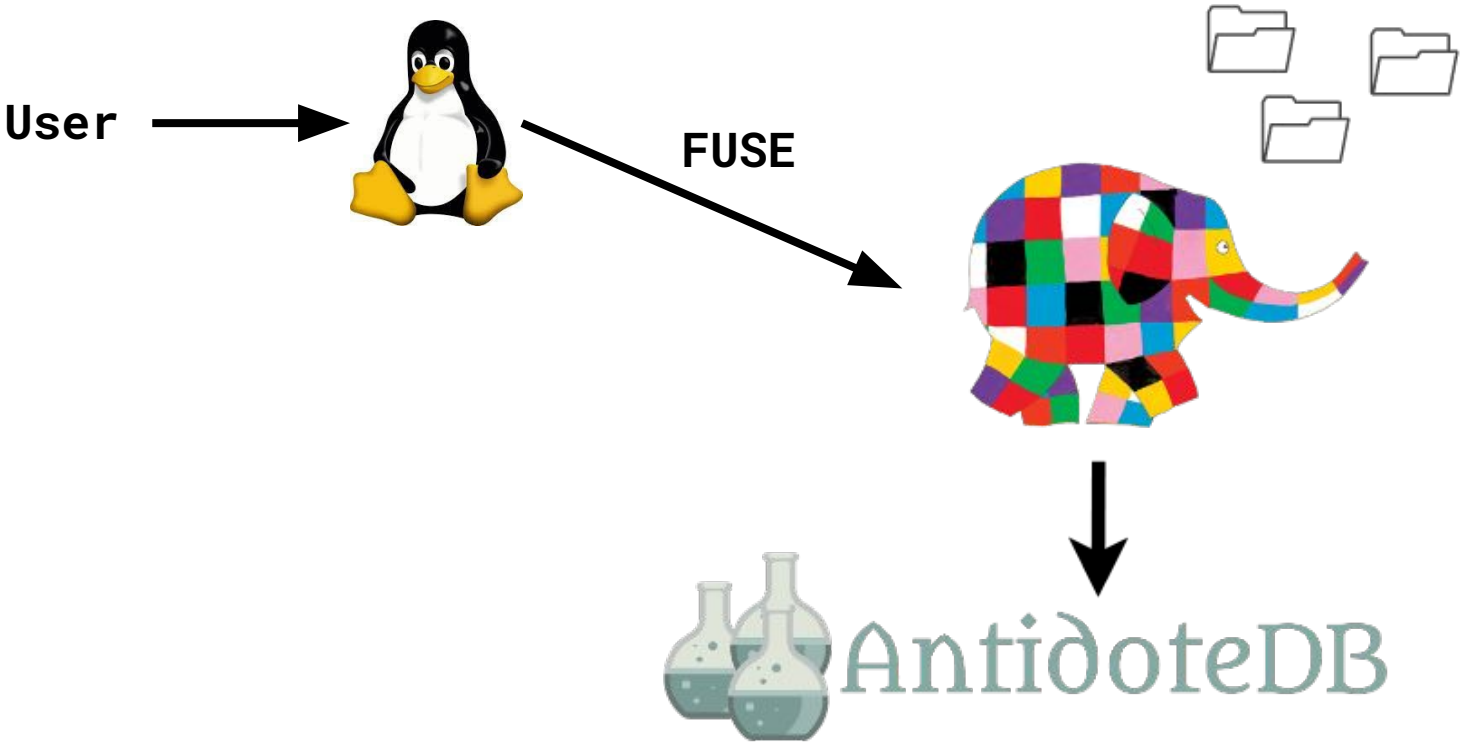
What is ElmerFS?

- POSIX file system interface.
- Active-Active geo-replication.
- Highly Available

CRDTs are a perfect fit for that!

- Independant and concurrent updates without coordination.
  Update can be accepted in any order, the system will always converge.
- Strong eventual consistency.
  The strongest form of eventual consistency
- Optimistic Replication
  Accept the operation locally, apply it to other nodes later

Architecture overview



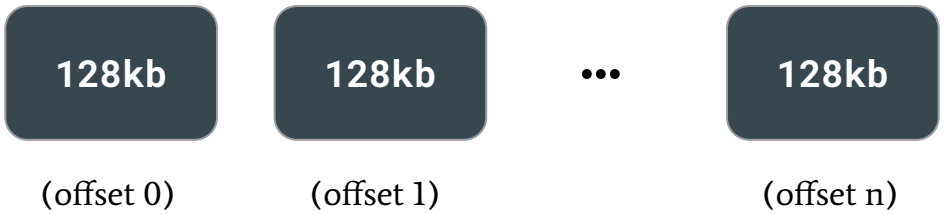
Directories

- Directories are represented as a set of tuples to reference inodes.
- Each request need to fetch the up to date directory with AntidoteDB.
- It can be expensive, we might download a huge folder for a single lookup.

Files

Files are stream of fixed size Last Writer Win Register addressed by their offset:

- No registry to reference those blobs.
- No Link between blobs
- Gaps are allowed



The other side of the coin

- **Name conflicts.**
- Divergent renames.
- Cyclic renames.
- Deletion of inodes.
- Content conflicts.

Alice and Bob are in a hurry.

```
Alice$ vim shared/report.md

                In the meantime...

  Bob$ emacs shared/report.md
```

What should happen ?

What existing systems are doing *

Cloud services	Strategy
Google Drive	Rename files (divergent)
One Drive	Rename files (consistent)
Dropbox	Rename files (consistent)

* Design and Implementation of a Concurrency Benchmark Tool for Cloud Storage Systems Weiwei Cai et al.

We can rename files!

```
$ ls /shared/
$ “report.md - (1)” “report.md - (2)”
```

You need to know how the system works to predict its behavior...

...and that the application didn’t create any conflicting files.

What we would like to happen

- A simple mental model.
- No after-the-fact corrections.
- Prevent applications from breaking.

Alice and Bob try ElmerFS.

Alice$** vim shared/report.md // **Bob$ emacs shared/report.md

Leads to

```
Alice$ ls /shared/
Alice$ “report.md” “report.md:Bob”
```

```
Bob$ ls /shared/
Bob$ “report.md” “report.md:Alice”
```

We can use a simple set right ?

We can represent directories as a set...

```
{ …, (name: “report.md”, ino: 0),
     (name: “report.md”, ino: 1), … }
```

But this does not solve the problem at all!

Convergence does not mean correctness!

Track the operation origin

We need to identify the origin of the operation:

```
{ …, (name: “report.md”, ino: 0, viewId: Alice),
     (name: “report.md”, ino: 1, viewId: Bob), … }
```

Every operation has a view ID associated with it.

Interfacing with Bob's obliviousness.

What the system sees:

```
{…, (name: "report.md", ino: 0, viewId: Bob), …}
```

What the system shows (implicit/explicit):

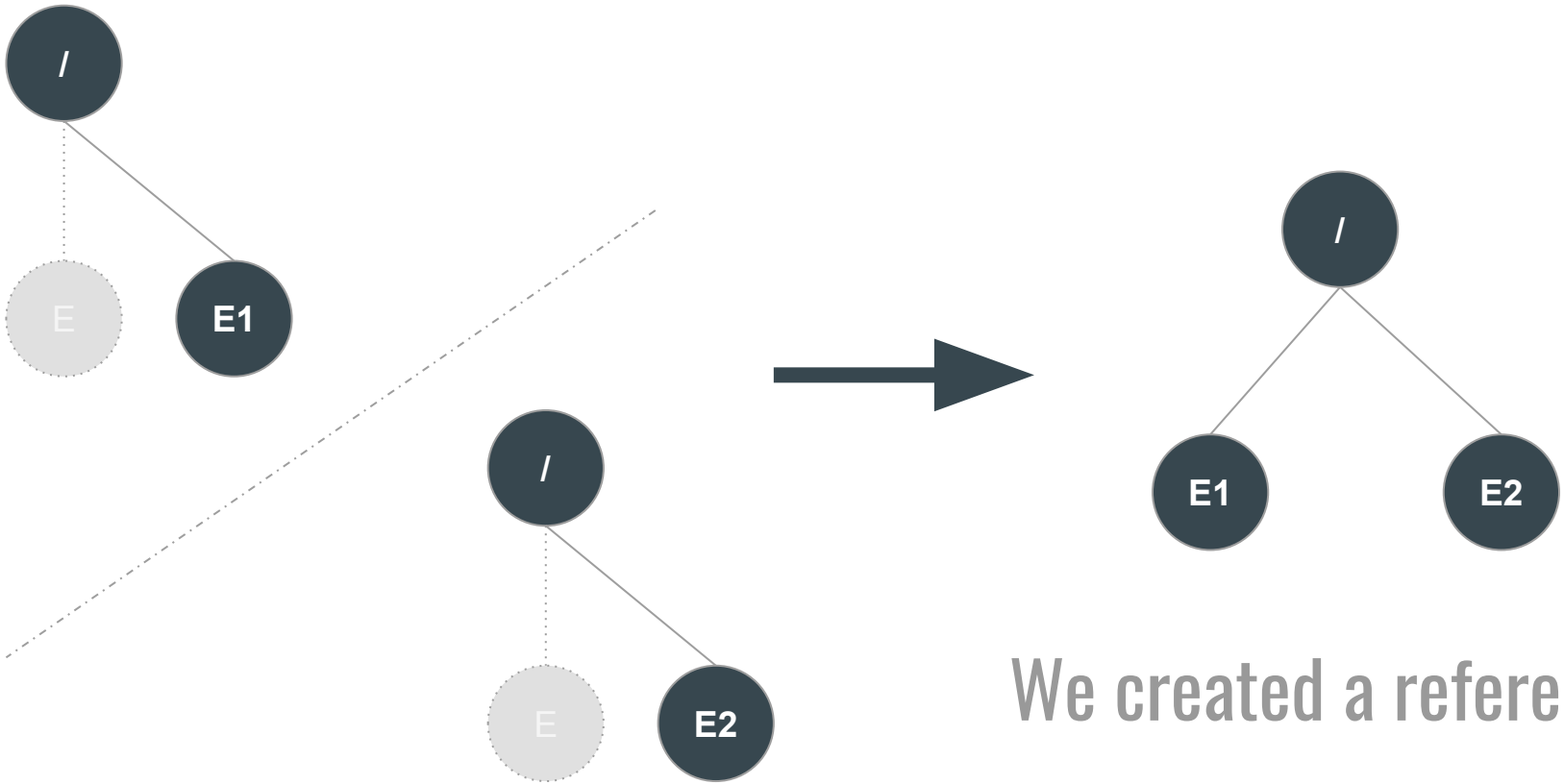
```
Bob$ ls shared/report.md
   $ report.md
```

```
Bob$ ls shared/report.md:Bob
   $ report.md
```

The other side of the coin

- Name conflicts.
- **Divergent renames.**
- Cyclic renames.
- Deletion of inodes.
- Content conflicts.

Divergent renames



We created a reference!

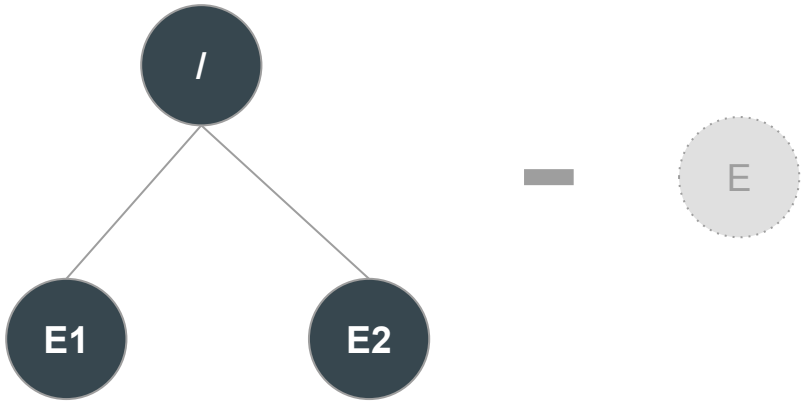
Reference counting doesn't work

- A rename operation only moves references.
- Uniqueness and transactions

  (parent_ino, ino, name, view_id) is unique, we keep them in a CRDT set.
- Use Last Writer Win semantic for folders

  To elect only one reference if POSIX compliance is necessary.

Divergent renames




```
{ (parent: “/”, name: “E1”, ino: 0, viewId: Bob),
  (parent: “/”, name: “E2”, ino: 0, viewId: Alice) }

    −   { (parent: “/”, name: “E”, ino: 0, viewId: Bob) }
```

The other side of the coin

- Name conflicts.
- Divergent renames.
- **Cyclic renames.**
- **Deletion of inodes.**
- **Content conflicts.**

Experiments

- How the filesystem handles Active-Active geo-distribution
- Compare it against a filesystem with strong consistency
- Use workload representative of a real-ish use case.

Choosing a contender

- There is no strict equivalent of ElmerFS.
- GlusterFS is a strongly consistent file system, open-source and well known .
- It support asynchronous Active-Passive / synchronous Active-Active GEO-Replication.

Choosing a workload

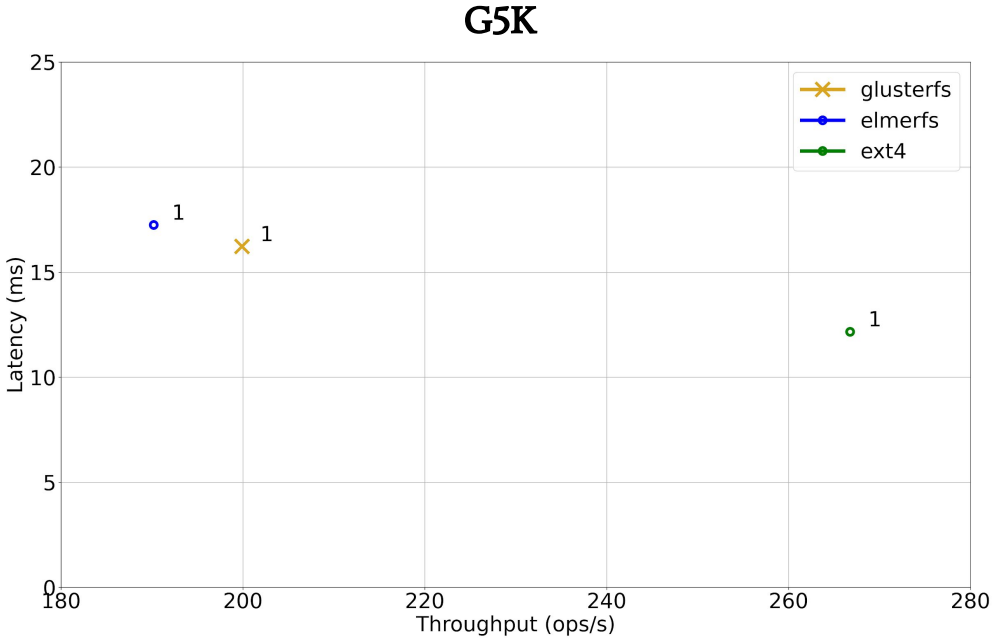
Experiments in 3 phases.

1. **One Node, Single DC:** Understanding the raw performance of typical workloads.

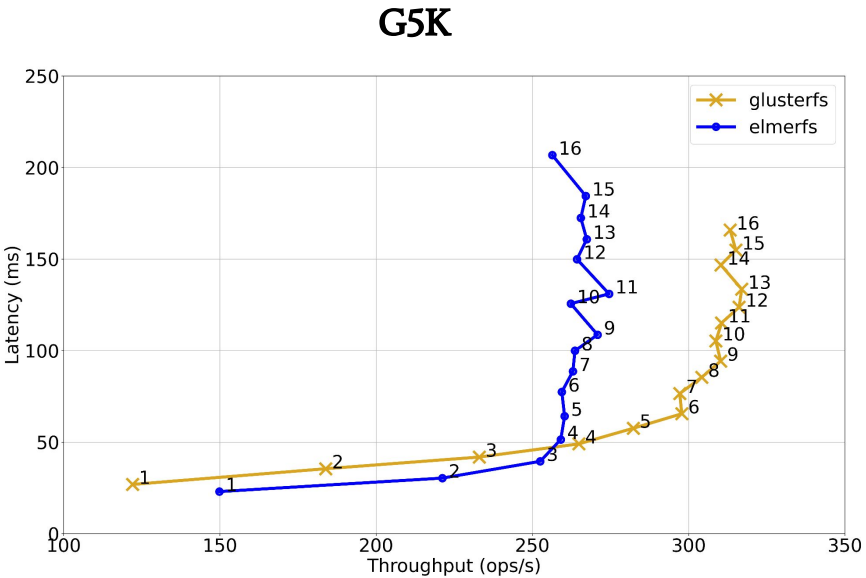
2. **Multi Node, Single DC**: Ensure that both ElmerFS and GlusterFS are well configured.

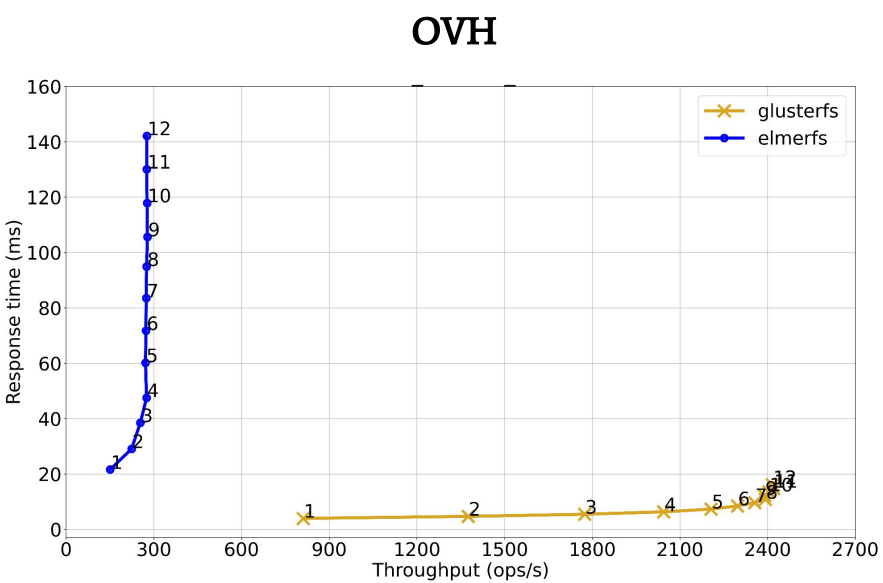
3. **Multi Node, Multi DC**: Measure response time in a GEO-replicated scenario to see the behaviour of each system in these conditions

Experiments - Phase 1

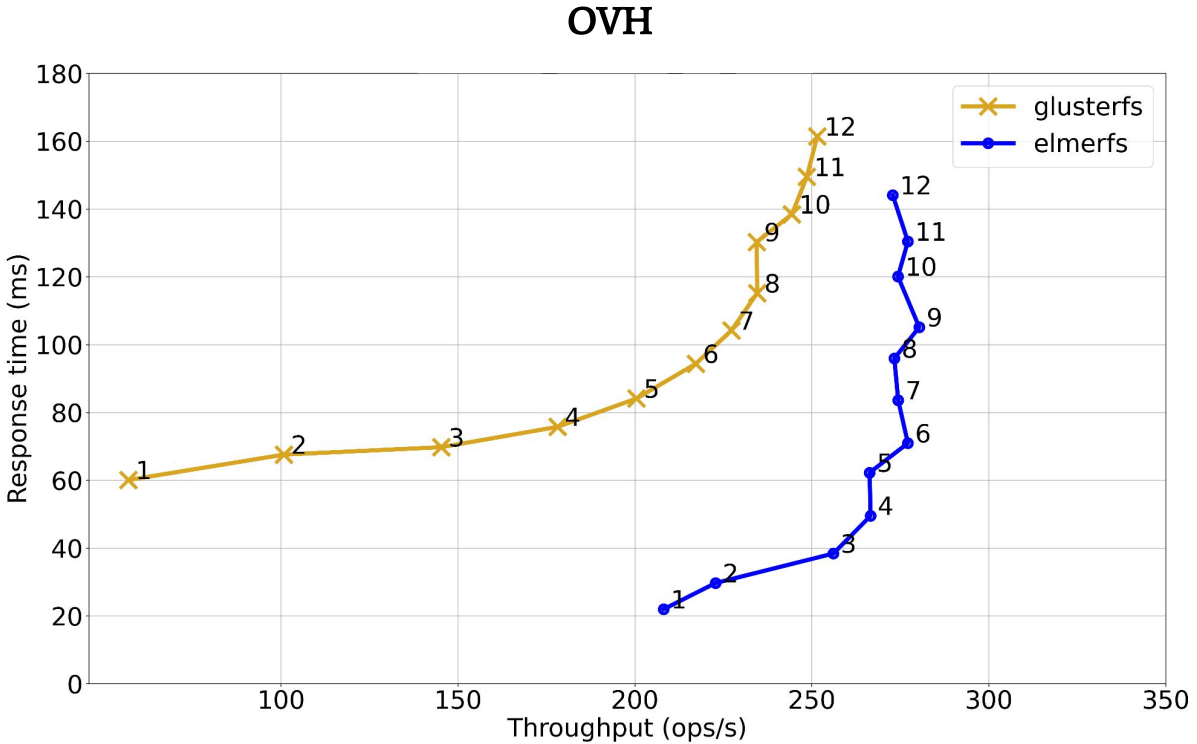


Experiments - Phase 2 - 1DC / 6 Nodes





Experiments - Phase 3 - 3DC / 6 Nodes



Implementations

- AntidoteDB
- ElmerFS
- Filebench
- Cloudal