CRDTs for truly concurrent file systems

•••

Romain Vaillant, Dimitrios Vasilas, Marc Shapiro,
Thuy Linh Nguyen

File systems

- A widely used solution for data sharing
- No longer limited to local uses
- Compatibility with legacy applications that expect POSIX

What is expected for these services

- Low response time.
- Always available.
- Scalable

The other side of the medal

- **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”
```

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.

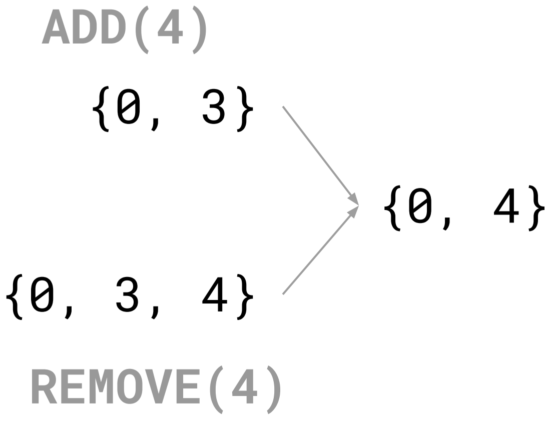
  The strongest form of eventual consistency

- Optimistic Replication

  Accept the operation locally, apply it to other nodes later

A simple CRDT: A Set.

Adds Win:



ADD(4)

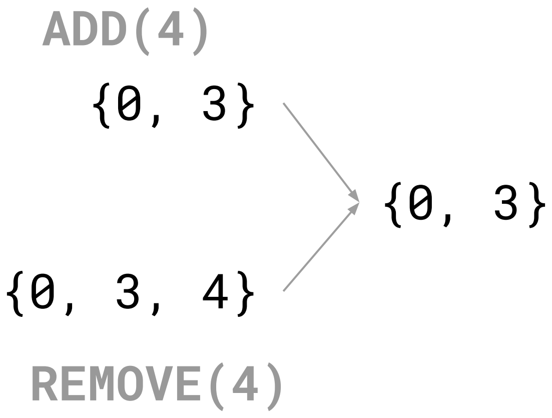
{0, 3}

{0, 3, 4}

REMOVE(4)

{0, 4}

Removes Win:



ADD(4)

{0, 3}

{0, 3, 4}

REMOVE(4)

{0, 3}

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):

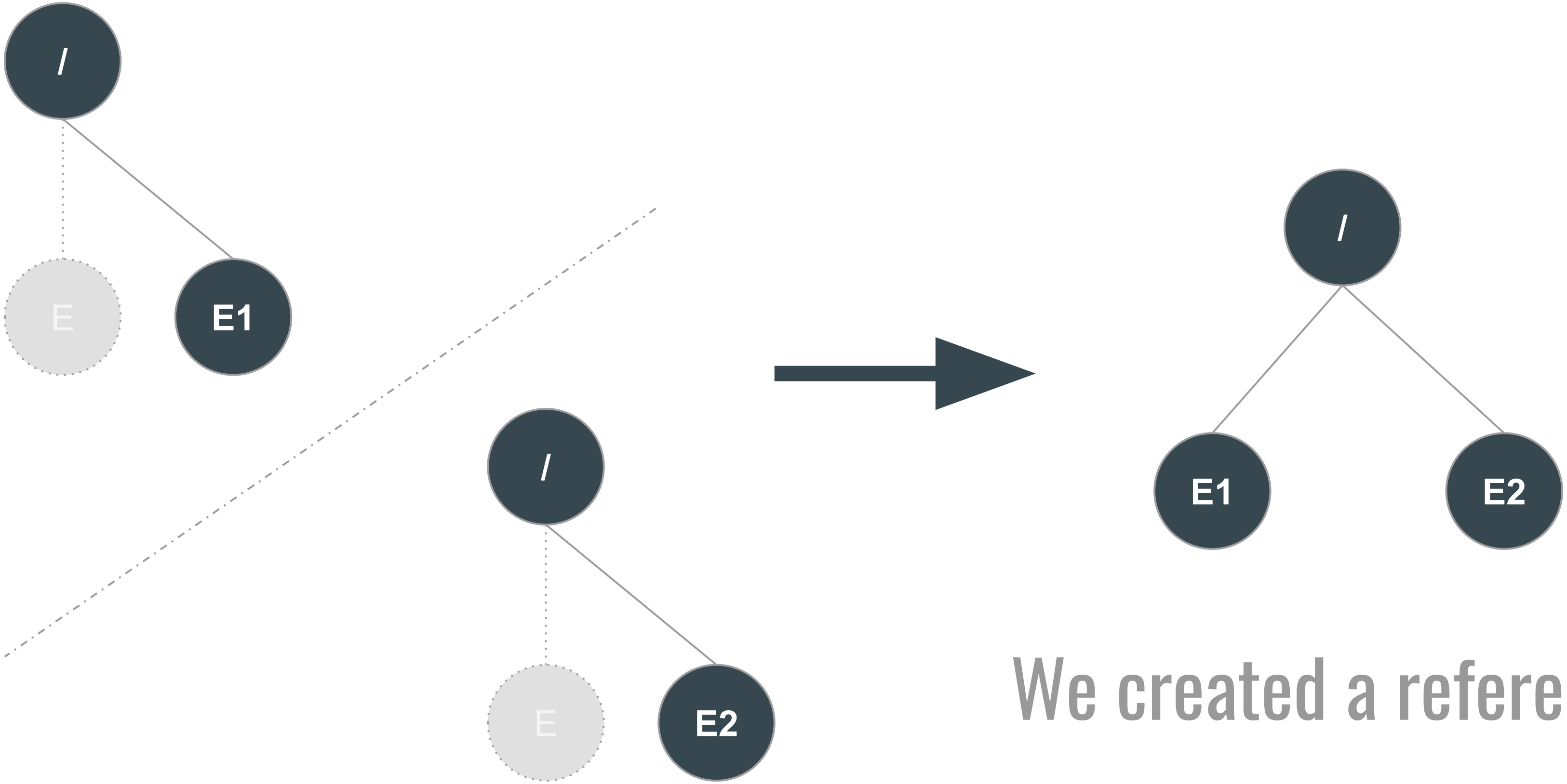
```
Bob$ ls shared/report.md
   $ report.md
```

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

The other side of the medal

- 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.

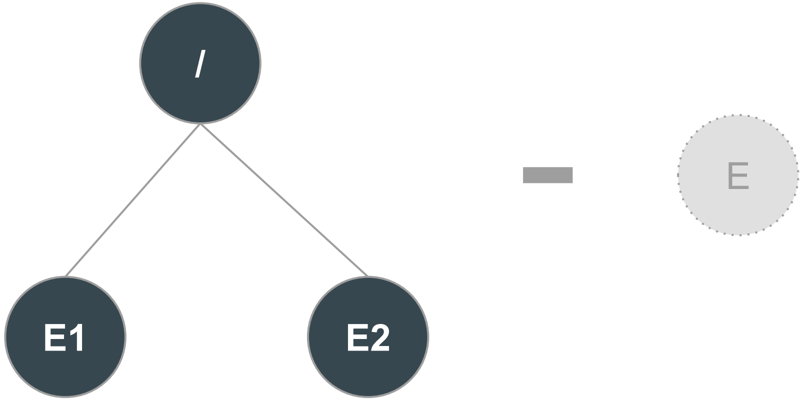
- Uniqueness and transactions

  (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 medal

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

Is this all theory ?

- AntidoteDB
- ElmerFS

Lesson learned.

- CRDT ensures that your system will converge

  But are not aware of the invariant of the application.

- The application designer must think on how operations interact

  To use the CRDTs properties to their advantages.

- The just right consistency.

  Only use synchronisation when strictly necessary.

Takeaways

- CRDTs properties are a good fit for geo-distributed file systems.
- Some problems remains: cycles, space reclamation…
- Experiments needed, on the interface and performance tradeoffs.

Thank You!

•••