goto;

GOTO
AARHUS 2023

Version control post-Git

#GOTOaar

Pierre-Étienne Meunier (Coturnix, Pijul)

Plan

goto;

Version control

Our solution

Implementation

Hosting platform: a new hope

What is version control?

goto;

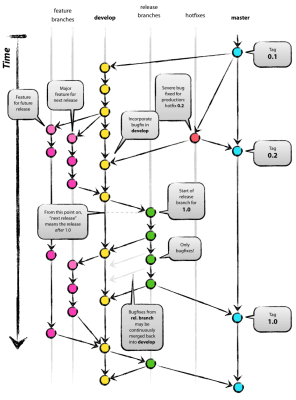
- One or more coauthors edit a tree of documents concurrently
- Asynchronous edits: coauthors can choose when they want to “sync” or “merge”
- Edits may **conflict**
- Review a project’s history

A solved problem?

Our tools (Git, Hg, SVN, CVS…):

- Aren't used by non-coders, despite their maturity (30 years+)
- Are almost unusable without a global central server (GitHub)
- Require strong work discipline and planning
- Waste significant human worktime at a global scale

Improvements have been proposed (Darcs) but don't really scale.



Note: in this talk we only consider open source version control systems

Our demands

goto;

We want:

- Associative merges:

  Changes A and B together are the same as A, followed by B.

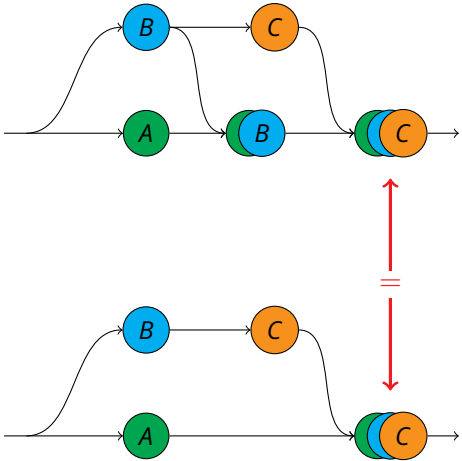
- Commutative merges:

  If A and B can *be produced* independently*, their order does not matter.*

- Branches (or not: more on that later)
- Low algorithmic complexity, and ideally fast implementations

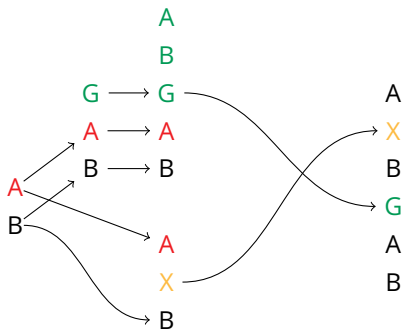
Associative merges, a.k.a “one-by-one review”

goto;

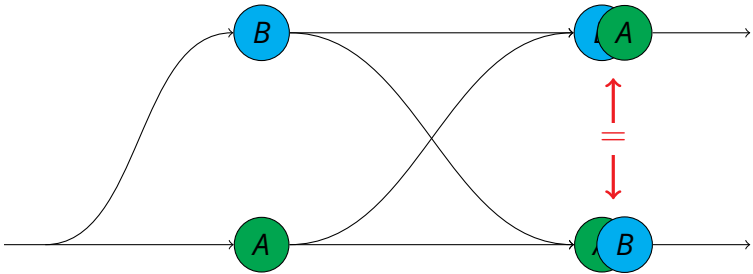


3-way merge (Git, Hg, SVN, CVS…) is not associative

goto;



Commutative merges



Commutative merges: why?

goto;

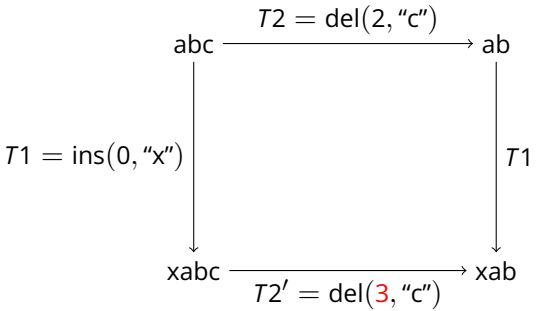
Git and SVN are **never** commutative, why would we want this?

- **Unapplying** old changes, even after others have been applied.
- **Cherry-picking**.
- **Partial clones**: pull the patches related to a subproject.

States vs changes

goto;

- Git, Hg, SVN, CVS… store **states**, and compute **changes** when needed (3-way merge).
- What if we did the **opposite**?
- What if we stored **both**?



- **Darcs** does this, and uses it to detect conflicts
- Quadratic explosion of cases
- A nightmare to implement

A hybrid (state/change) approach: CRDTs

goto;

- General principle: design a structure where all operations have the properties we want
- Natural examples: increment-only counters, insert-only sets…
- More subtle: tombstones, Lamport clocks…
- Useless: a full Git repository (not just `HEAD`)

Plan

goto;

Version control

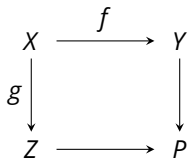
Our solution

Implementation

Hosting platform: a new hope

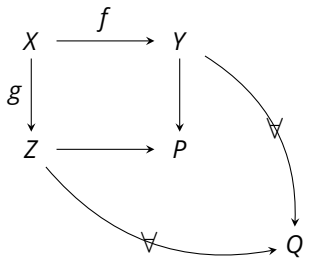
- Where we need a good tool the most
- The exact definition depends on the tool
- **Example:** Alice and Bob write to the same file at the same place
- **Example:** Alice renames a file from f to g while Bob renames f to h
- **Example:** Alice renames a function f while Bob adds a call to f

For any two patches f and g, we want a unique state P such that:



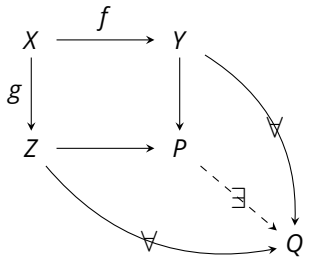
Started by Samuel Mimram and Cinzia Di Giusto

For any two patches f and g, we want a unique state P such that:
For any state Q accessible by Alice and Bob after f and g, respectively



Started by Samuel Mimram and Cinzia Di Giusto

For any two patches f and g, we want a unique state P such that:
For any state Q accessible by Alice and Bob after f and g, respectively
There is a patch from P to Q.

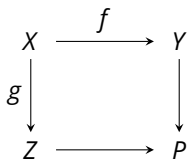


If P exists (implying uniqueness), we call P the **pushout** of f and g.

Started by Samuel Mimram and Cinzia Di Giusto

Problem: the pushout doesn't always exist

- Equivalent to saying that conflicts happen.
- How to generalise the representation of states (X, Y, Z) so that all pairs of changes (f and g) have a pushout?

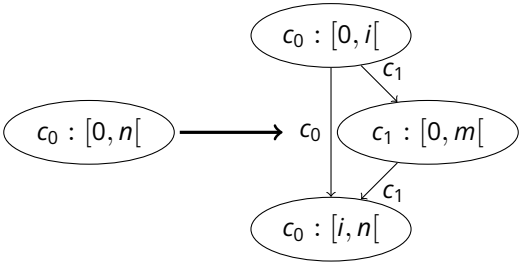


Solution: States are directed graphs, where:

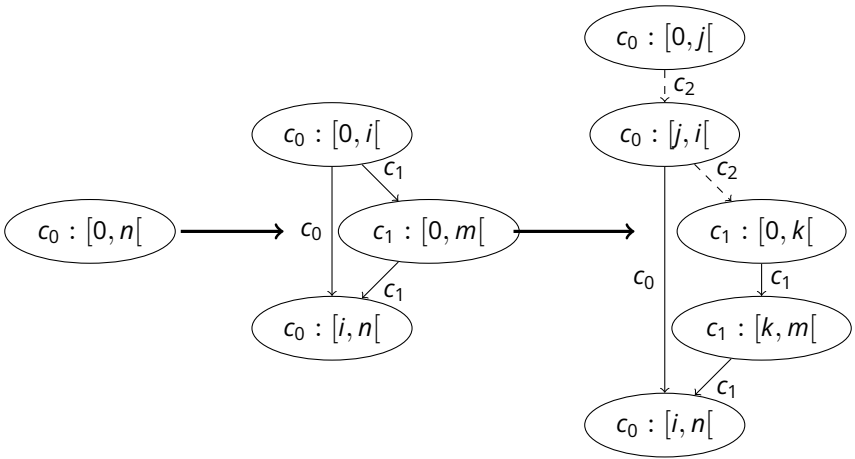
- Vertices are bytes (or byte intervals).
- Edges represent the union of all known orders between bytes.

- Vertices are labelled by a change number c_0 and an interval (such as $[0, n[$) in that change.
- Edges are labelled by the change that introduced them.

Here, c_1 adds m bytes between positions $i - 1$ and i of c_0:



Deleting bytes j to i from c_0, and 0 to k from c_1:



That’s all we need!

goto;

Two kinds of changes:

- Add a vertex, in a **context** (parents and children)
- Change an edge’s label

Our definition of conflicts

goto;

- **Alive** vertices are vertices whose incoming edges are all alive.
- **Dead** vertices are vertices whose incoming edges are all dead.
- Other vertices are called **zombies**.

A graph has **no conflict** if and only if it has no zombie and all its alive vertices are totally ordered.

Some remarks

- Changes are partially ordered by their dependencies on other changes.
- Cherry-picking is the same as applying a patch.
- No `git rerere`: conflicts are solved by changes, which can be cherry-picked.
- Partial clones/monorepos: easy as long as “wide” patches are disallowed.
- Large files: we only need the description of operations (insertions/deletions).

Plan

goto;

Version control

Our solution

Implementation

Hosting platform: a new hope

Working with large graphs on disk

goto;

- We can't load the entire graph each time.
- Store edges in a key-value store.
- Transactions: passive crash-safety.
- Branches: efficiently forkable store.

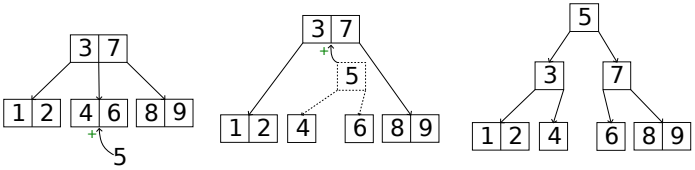
Introducing **Sanakirja**, an on-disk transactional KV store

goto;

- File block allocator
- Crash-safety using referential transparency and copy-on-write.
- Forkable in $O(\log n)$, where n is the total size.
- Written in Rust (but with a tricky API).
- Generic underlying storage layer.

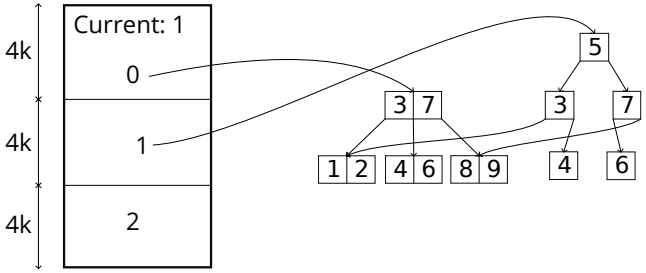
Crash safe using multiple B trees and roots

goto;



Crash safe using multiple B trees and roots

goto;

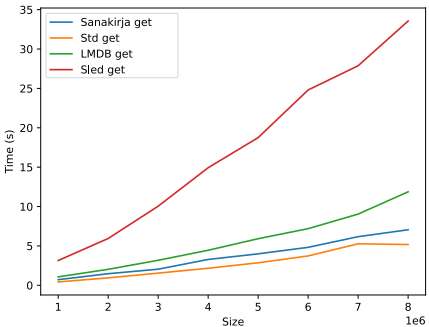


- Updating the “current” (first 8 bytes of the file) commits the next version.
- Writers don’t block readers!

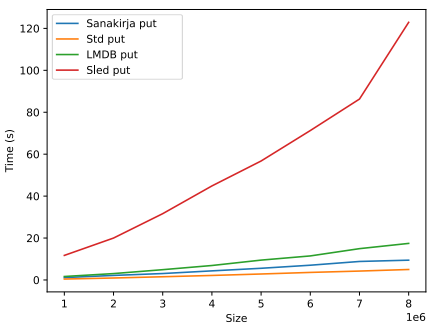
Sanakirja is the fastest we've tested

- Performance of retrieval (get) and insertion (put) into a B tree.
- Not specific to Pijul (but long values not yet implemented).

Get



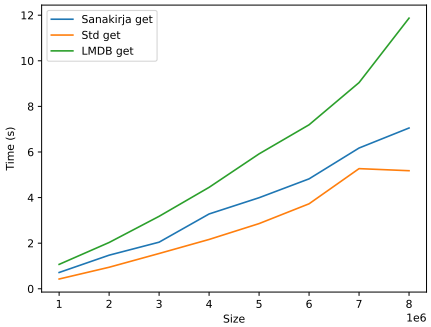
Put



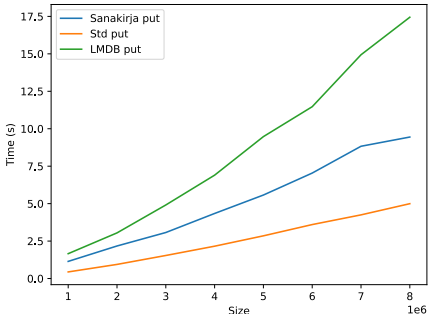
Sanakirja is the fastest we've tested

- Performance of retrieval (get) and insertion (put) into a B tree.
- Not specific to Pijul (but long values not yet implemented).

Get



Put



Plan

goto;

Version control

Our solution

Implementation

Hosting platform: a new hope

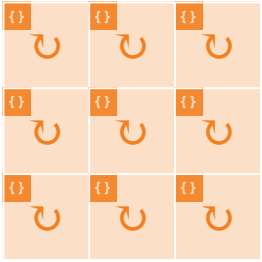
Building a hosting platform with a team of one

goto;

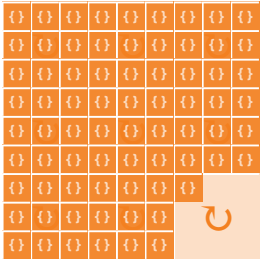
- First version released in 2016, Rust code + PostgreSQL running on a single machine
- OVH Strasbourg data center fire in March 2021.
- Now: replicated setup using Pijul-as-a-CRDT, and Raft to replicate Postgres

Main issue: high loads → Postgres failures → switchovers → **data loss**

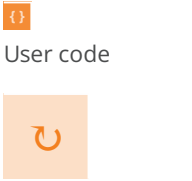
Using function-as-a-service



Traditional architecture



Workers V8 isolates



Taken from the Cloudflare documentation

Using function-as-a-service

goto;

```
export default {
  async fetch(request) {
    return new Response('Hello worker!', { status: 200 });
  },
};
```

Taken from the Cloudflare documentation

A robust solution with FaaS

- Can we run (or simulate) a Pijul repository in a pure function-as-a-service framework?
- Main challenge: high latency, eventually consistent storage.
- Compiling Sanakirja to WASM, storing pseudo-memory pages on the storage engine.

A robust solution with FaaS

goto;

- Can we run (or simulate) a Pijul repository in a pure function-as-a-service framework?
- Main challenge: high latency, eventually consistent storage.
- Compiling Sanakirja to WASM, storing pseudo-memory pages on the storage engine.
- Using the multiple heads to deal with eventual consistency.

A robust solution with FaaS

goto;

- Can we run (or simulate) a Pijul repository in a pure function-as-a-service framework?
- Main challenge: high latency, eventually consistent storage.
- Compiling Sanakirja to WASM, storing pseudo-memory pages on the storage engine.
- Using the multiple heads to deal with eventual consistency.
- We don't need a full Pijul:
  Checking dependencies and maintaining a list of patches is enough

Technical details

goto;

- Typescript for web parts
- Svelte for the UI
- Rust/WASM for the Pijul parts
- Can be self-hosted using Cloudflare's `workerd`
- Open source (AGPL), released progressively, starting **today!**

https://nest.pijul.org

Conclusion

goto;

- Open Source version control based on proper algorithms.
- Scalable to monorepos and large files.
- Potentially usable by non-coders: parliaments, artists, lawyers, Sonic Pi composers, LEGO builders…
- Hosting service available since today.
- Personal note: doing many things at the same time never works, until it does.

Acknowledgements: Florent Becker, Tankf33der, Rohan Hart, Chris Bailey, Angus Finch…

goto;

Thanks for your attention

goto;

Don’t forget to
vote for this session
in the **GOTO Guide app**