Version control in the age of distributed computing

Pierre-Étienne Meunier

March 17th, 2023

Version control as a distributed system

- One or more coauthor editing a shared datastructure (e.g. a file)
- Applying **changes** (or patches) to a common version
- Sometimes, these changes **conflict**, and conflicts must be **resolved**
- Another feature : reviewing and changing the project's history
- Fundamental questions $\neq$ important/useful

Conflicts

- Where we need a good tool the most
- The exact definition depends on the tool
- **Example :** Alice and Bob write at the same place in the same file
- **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*

Some (minimal) bibliography

- The CAP theorem (Brewer, 1998) : a system robust to network partitions cannot be **consistent** and **available** at the same time.

- Choosing **consistency** : leader election

  Paxos (Lamport 1989), Raft (Ongaro, Ousterhout 2011)

- Choosing **availability** :

  Operational Transforms, or OTs (Ellis, Gibbs 1989)

  Conflict-free Replicated DataTypes, or CRDTs (Shapiro et al 2011)

Desirable properties of changes

1. **Associativity :** changes can be applied one by one or together, i.e. $(AB)C = A(BC)$
2. **Commutativity :** changes that could be written independently can be applied in any order, i.e. $AB = BA$
3. Changes can be **unapplied** even after other independent changes have been added.

Trying to simulate algebraic properties

Git, Mercurial, SVN, CVS... try to simulate some of these properties :

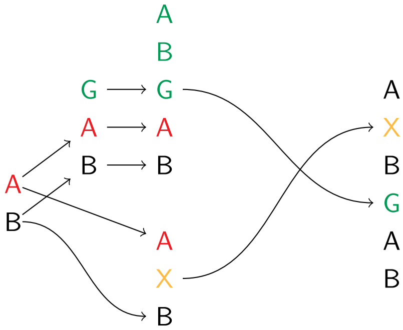
- `git merge` tries to be associative, $(AB)C = A(BC)$
  Long-lived branches are bad practice.

- `git cherry-pick` tries to simulate commutativity, $AB = BA$
  Don't merge that same branch later on or you'll also get unexplained conflicts.

Some tools focus on **conflicts** (`git rerere`, `jujutsu`).

We want to focus on **their causes** instead.

Git and SVN are not associative



Towards a suitable datastructure

For any two patches f and g, we would like a state P to exist and be unique, such that :

$$\begin{tikzcd}
X \arrow[r, "f"] \arrow[d, "g"'] & Y \arrow[d] \\
Z \arrow[r] & P
\end{tikzcd}$$

Work started by Samuel Mimram (École Polytechnique)

Towards a suitable datastructure

For any two patches f and g, we would like a state P to exist and be unique, such that :
For any state Q accessible by Alice and by Bob after f and g

$$\begin{tikzcd}
X \arrow[r, "f"] \arrow[d, "g"'] & Y \arrow[d] \arrow[ddr, bend left, "\forall"] & \\
Z \arrow[r] \arrow[drr, bend right, "\forall"'] & P & \\
& & Q
\end{tikzcd}$$

Work started by Samuel Mimram (École Polytechnique)

Towards a suitable datastructure

For any two patches f and g, we would like a state P to exist and be unique, such that :
For any state Q accessible by Alice and by Bob after f and g
There exists a path from P to Q.

$$\begin{tikzcd}
X \arrow[r, "f"] \arrow[d, "g"'] & Y \arrow[d] \arrow[ddr, bend left, "\forall"] & \\
Z \arrow[r] \arrow[drr, bend right, "\forall"'] & P \arrow[dr, dashed, "\exists"] & \\
& & Q
\end{tikzcd}$$

If P exists and is unique, P is called the *pushout* of f and g.

Work started by Samuel Mimram (École Polytechnique)

Problem : pushouts don't always exist

- Or otherwise said : sometimes, there are conflicts
- How to generalise the representation of states (like X, Y, Z) so that all pairs (like (f, g)) have a (unique) pushout ?

$$\begin{tikzcd}
X \arrow[r, "f"] \arrow[d, "g"'] & Y \arrow[d] \\
Z \arrow[r] & P
\end{tikzcd}$$

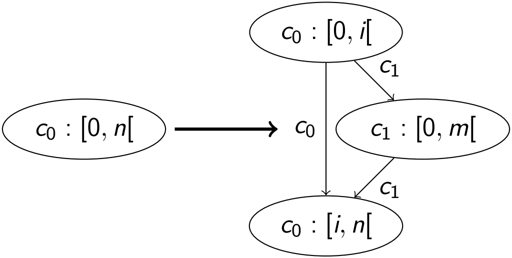
Solution : States are directed graphs where :

- Vertices are bytes (or byte intervals).
- Edges are the union of all orders known between vertices ("this byte comes before that byte").

Adding a line

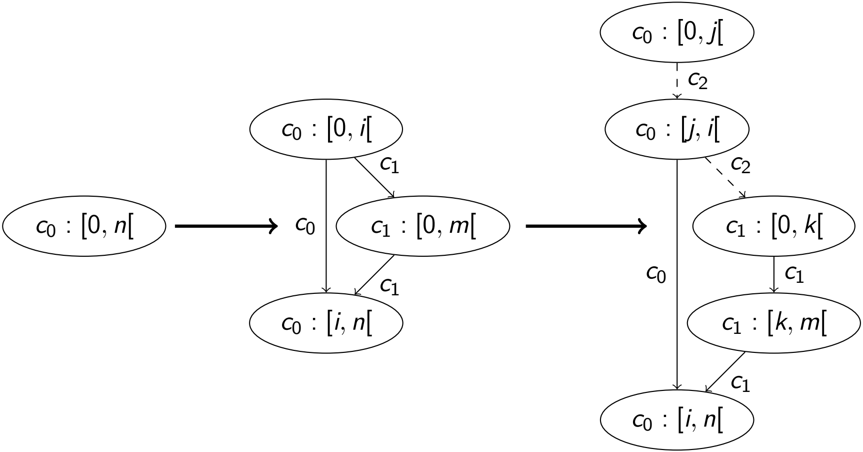
- Vertices are labelled by a patch identity (example : c_0) and an interval of bytes (example : $[0, n[$) within that patch.
- Edges are labelled by the patch that introduced them.

Here is patch c_1 adding m bytes between positions $i-1$ and i of patch c_0 :



Deleting a line

Patch c_2 deletes bytes j to i (excluded) from c_0, and then 0 to k (excluded) from c_1 :

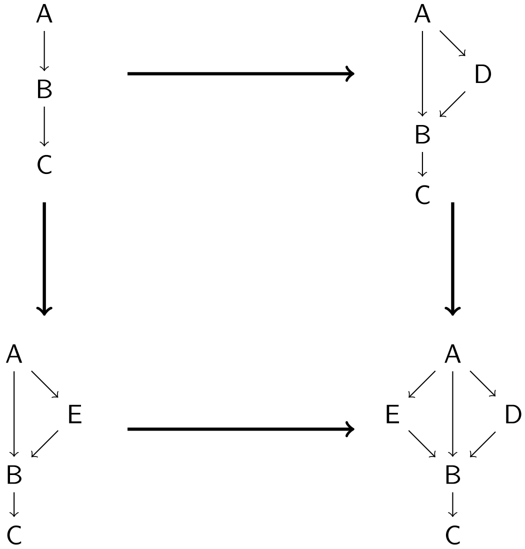


Conflicts

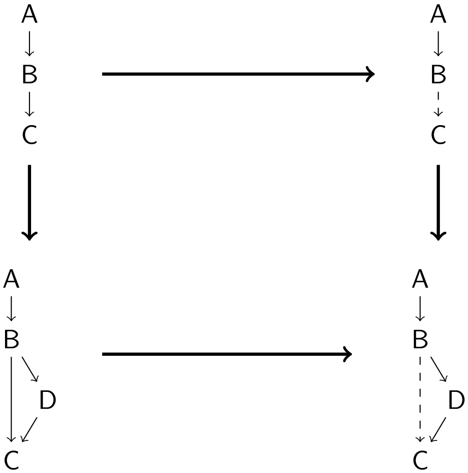
- **Alive vertices** are vertices whose incoming edges are all alive
- **Dead vertices** are vertices whose incoming edges are all dead
- Other vertices are called **zombies**.

A graph is **conflict-free** iff it has no zombies
and all alive vertices are totally ordered.

Example : an order conflict



Example : deleted down context



Implementation of Pijul

- Almost entirely written in Rust
- Beta since January 18th, 2022
- Notable components :
  - Sanakirja : a library for transactional, on-disk datastructures, with a central basic structure (forkable B trees)
  - Libpijul : the algorithms described in this talk
  - Pijul : command-line interface, network ops

Quick zoom in on Sanakirja

- **Forkable in near-constant time**, transactional, on-disk KV store
- Fastest open source library for all supported operations
- Reusable for other datastructures, not necessarily search trees
- Extensible to non-disk backends like serverless
- Too generic = hard to use (contributions wanted !)

Important implementation details

- Cherry-picking, partial clones, repository merges don't need any special treatment.
- Patches are detachable from their contents : dead byte intervals don't need to be downloaded.
- Pijul has a generic backend, which can be used on disk, compressed files, serverless databases...

First attempt at hosting : nest.pijul.com

- Written in asynchronous Rust (Tokio + Hyper)
- Deployed with Nix + custom tools to OVH (OpenStack)
- Replicated in Canada, France and Singapore
- Motivated the creation of an SSH library to write the server
- Pijul repositories are particularly suitable for replication

First attempt at hosting : nest.pijul.com

Main issues :

- Went through OVH fire in Strasbourg in March 2021 → replication !
- Machines were hard to provision (for financial reasons + NixOS)
- Needed local geographical replicas of PostgreSQL, plus a leader.
  We sometimes lose data during leader switchovers
- Single-person team

Announcing the new Nest

- Typescript + some WASM, running on Cloudflare Workers
- Fake Pijul repositories, with Sanakirja running on top of Cloudflare KV or DO.
- Many small independent workers, much easier to contribute to
- Self-hosting possible (but not yet easy)

What's next ?

- **Open-source and funding :** nobody has ever been working full-time on this
- **Just like functional programming**, radical changes take time to be adopted
- Using version control **where Git cannot go** : video games, legal documents, participatory democracy...

What's next ?

- **Open-source and funding :** nobody has ever been working full-time on this
- **Just like functional programming**, radical changes take time to be adopted
- Using version control **where Git cannot go** : video games, legal documents, participatory democracy...

Thanks for your attention

(and sorry I couldn't be with you today !)