Irmin: a Git-like database library

Thomas Gazagnaire, Amir Chaudhry, Jon Crowcroft, Anil Madhavapeddy, Richard Mortier[1] David Scott[2] David Sheets and Gregory Tsipenyuk

University of Cambridge, University of Nottingham[1], Citrix Systems[2]

OCaml Workshop, 05/09/2014

Version Controlled Database

What if you could version control your database?

- See the history of updates using: `git log`
- Get the latest updates from X: `git pull X`
- Modify the database atomically: `git commit -a`
- Create a transaction: `git checkout -b tXXX`
- Share your local updates: `git push`

Yes, but what about merges?

- The nightmare of every Git user:

```
$ git merge X
Auto-merging <PATH>
CONFLICT (content): Merge conflict in <PATH>
Automatic merge failed; fix conflicts and then
commit the result.
```

Yes, but what about merges?

Can we resolve and deal with conflicts progammatically?

Irmin Approach

- The data in the database has a structure (ie. a *type*)
- The merge functions are *defined by the user*
- Having an history (`git log`) helps *a lot*:
    - 3-way merge
    - vs. CRDT (enriched state 2-way merges)

Example

Consider distributed counters

- type: `int`
- how do you merge the values 8 and 6?

Example

Consider distributed counters

- type: `int`
- how do you merge the values 8 and 6?
- knowing that the latest common value was 5?

Example

Consider distributed counters

- type: `int`
- how do you merge the values 8 and 6?
- knowing that the latest common value was 5?
- Answer: $5 + (8 - 5) + (6 - 5) = 9$

Irmin

- We've played with these ideas in a complete implementation in pure OCaml: `https://github.com/mirage/irmin`
- We have a model of the OCaml heap with various backend implementations
  - `Obj` backend: no persistence, but no performance cost
  - `Git` backend: pure implementation of the Git protocoal, bi-directional interactions
- We have implemented various persistent datastructures with merge function
  - prefix trees
  - mergeable queues
  - mergeable ropes