

Weakly Consistent and Disconnected Operation

Tom Anderson (h/t Ray Cheng)

Why Weak/Disconnected?

File synchronization across users / devices
 Dropbox, metasync: data updated continuously

Source code control
 Git: data updated locally, explicit merges

Disconnected / intermittent connectivity
 Laptop and mobile apps, 3rd world: data updated locally, merged when connectivity is available

Serializability Recap

Serializability: everyone sees same read/write order: cache coherence, Paxos

Release consistency: reads/writes forced to complete at memory barriers, lock/unlock

Need a different model for concurrent updates and disconnected operation: always available writes

Background Reading

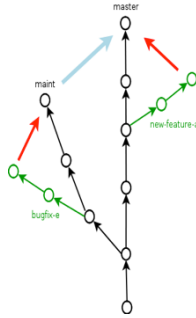
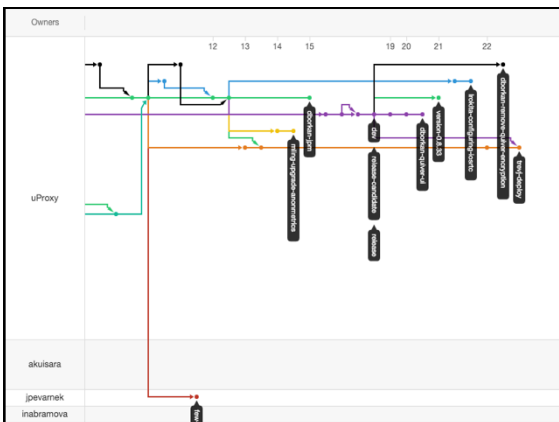
Terry et al. (1995)
[Managing Update Conflicts in Bayou, a Weakly Connected Replicated Storage System](#)

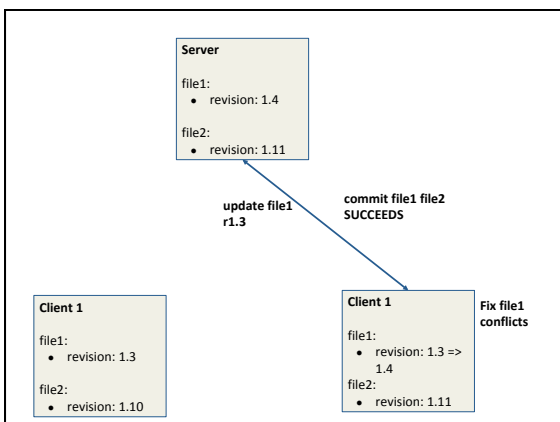
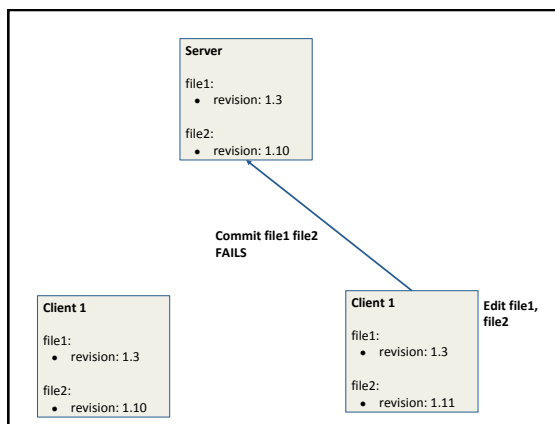
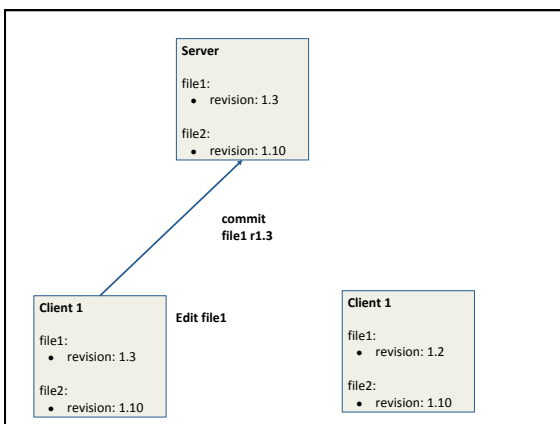
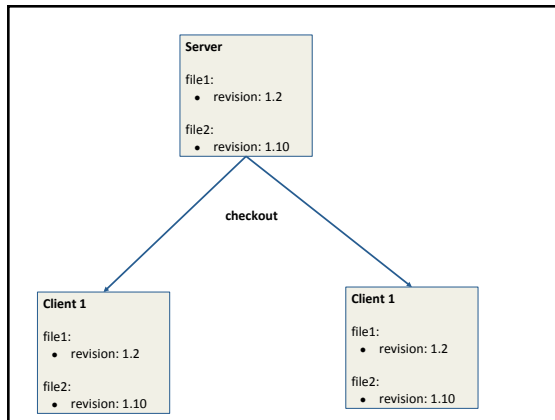
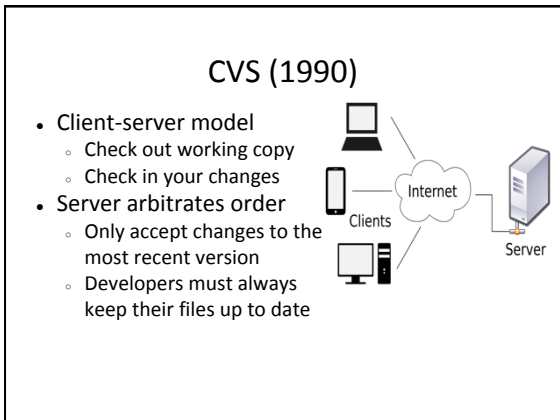
Xerox PARC project to build the first practical PDAs

Collaborative apps with partial connectivity

Source Code Control

- Eventual Consistency
 - Read/write local copy
 - Fix conflicts later
- Track history (with metadata)
- Concurrent editing / Many contributors
- Working copy: files don't change beneath you
 - Push / Pull to server/peers
 - Contributors may be offline / disconnected



- ### CVS Limitations
- Everyone edits the same repository
 - How does a subgroup implement a complex feature?
 - No local version control
 - cvs commit ~ git commit && git push
 - No log/ time travel
 - No versioning of moving / renaming files
 - Depends on live server to operate
 - Scaled / backed up / reachable
 - Branches were expensive
 - Updates not atomic (!)

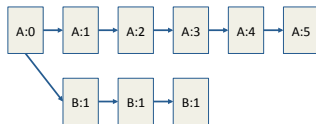
Apache SVN (2000)

- Improvements
 - Atomic commits
 - Renamed / moved / copied files retain version history
 - Versioning of directories and metadata
 - Cheap branches / tagging
- Centralized - server/client architecture
- Still active
 - All of Facebook's source code was in a single SVN repository until 2014

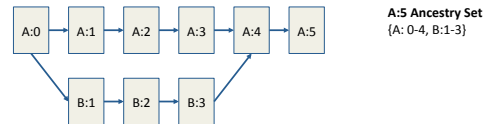
Commit Log



Branching



Merging



Conflicting updates detected with vector clocks
What then?

Merge Conflicts

Easy: create/delete/rename different files in directory => union of changes

Medium: changes to different lines of text file => diff+patch
Change to file that has been renamed => apply

Hard: changes to the same line of C source => ask user to fix

Another option: operational transforms

Merging and Causal Ordering

Operations that potentially are causally related are seen by every node of the system in the same order

Example:

C1: f=1 -> C2

C2: f=2 -> C3

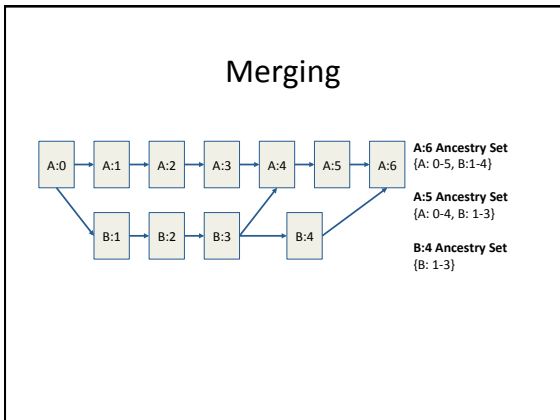
C3: f=3 -> C1

Example:

C1: a=1 -> C2

C2: b=2 -> C3

C3: c=3 -> C1

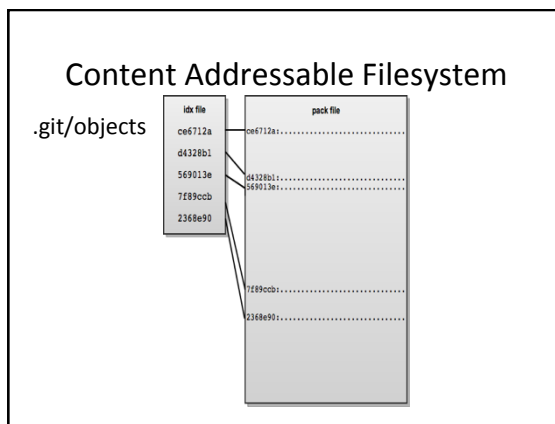


git (2005)

- Distributed!
 - Everyone is a replica
- Consistency and performance
 - Protects from memory, disk corruption
- Cheap branches / merges
- .git/
 - Config
 - Content-addressable filesystem
 - Log of changes (commit history)

Logs (Commit Histories)

- Complete log of changes (needed for time travel with source code control)
 - Directed acyclic graphs (DAG)
- commit
 - parents
 - deltas (changes to content)
 - hash - for consistency
 - metadata



Git Example

```

$ git init
$ echo "version 1" > test.txt
$ git add test.txt
$ git commit -m "first commit"

$ echo "version 2" > test.txt
$ echo "new file" > new.txt
$ git add ./
$ git commit -m "second commit"

$ mkdir bak
$ echo "version 1" > bak/test.txt
$ git add bak/
$ git commit -m "third commit"
    
```