UiT

THE ARCTIC UNIVERSITY OF NORWAY

Scalability of Distributed Version Control Systems

Michael J. Murphy, John Markus Bjørndalen, Otto J. Anshus

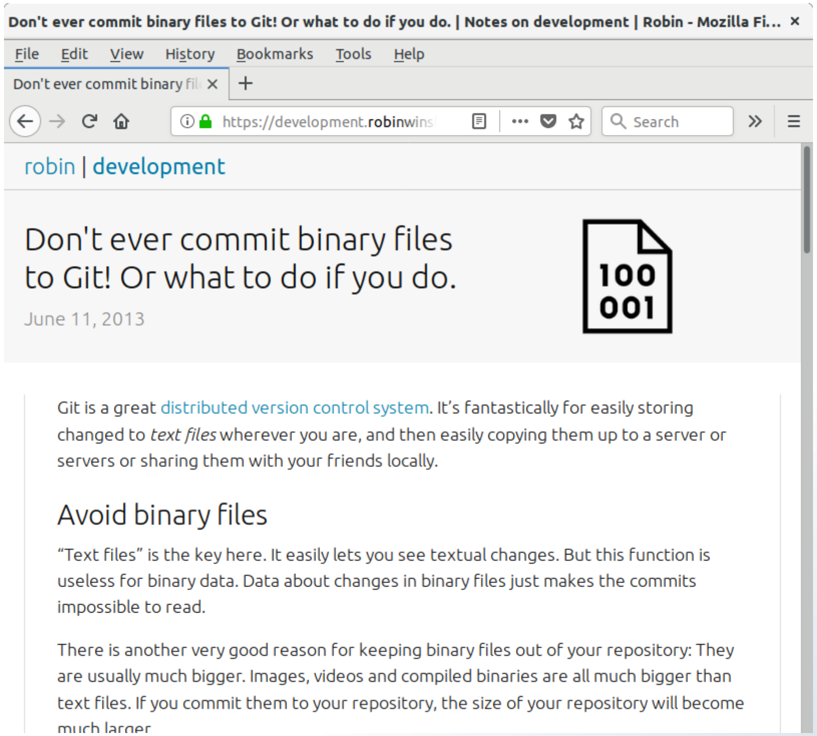
University of Tromsø

UNIVERSITY OF TROMSØ

Distributed Version Control

- Examples: Git, Mercurial
- Useful tools for data synchronization
- Distributed systems
- CAP theorem: focus on availability
- Do they scale?
- Why not?
- Experiments with large files and many files
- Prototype

Scalability?



- Designed for source code
- Text files, kilobytes
- Line-based diff for compression

- Trouble with binaries
- No sub-file compression
- Repo size keeps growing

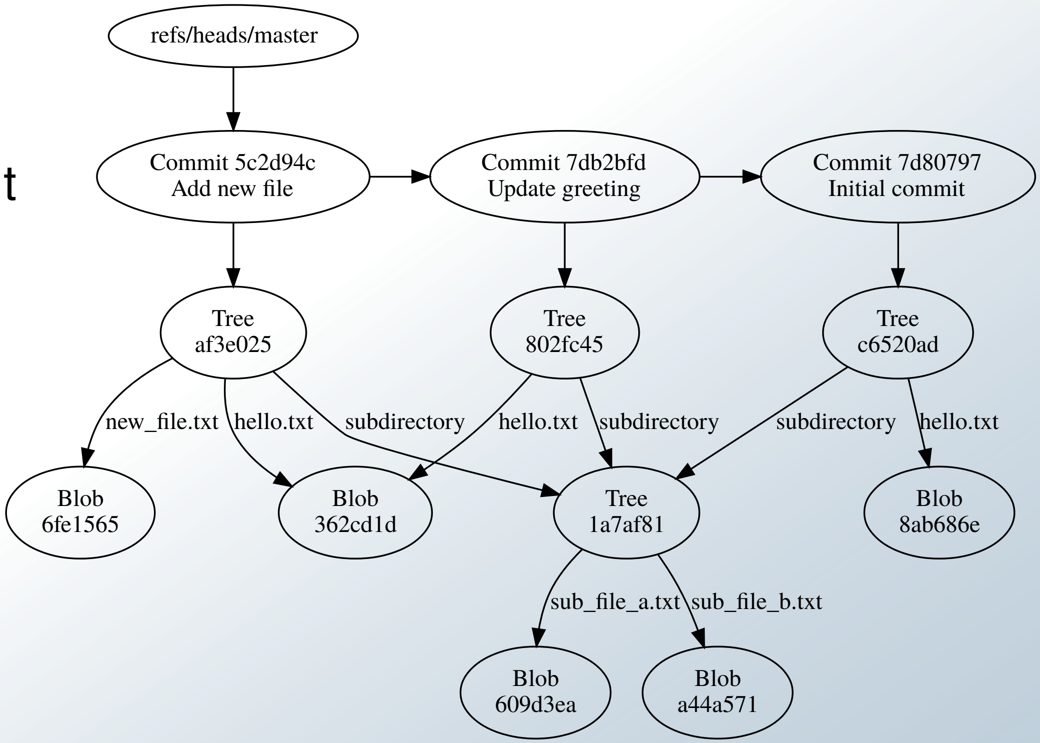
Efforts to Manage Large Files in Git

- Git-annex, Git-media, Git Large File Storage (Git LFS)
- All store the file somewhere else, with a pointer in the repo

- Boar, Bup
- Use Rsync rolling hash algorithm to break files into chunks

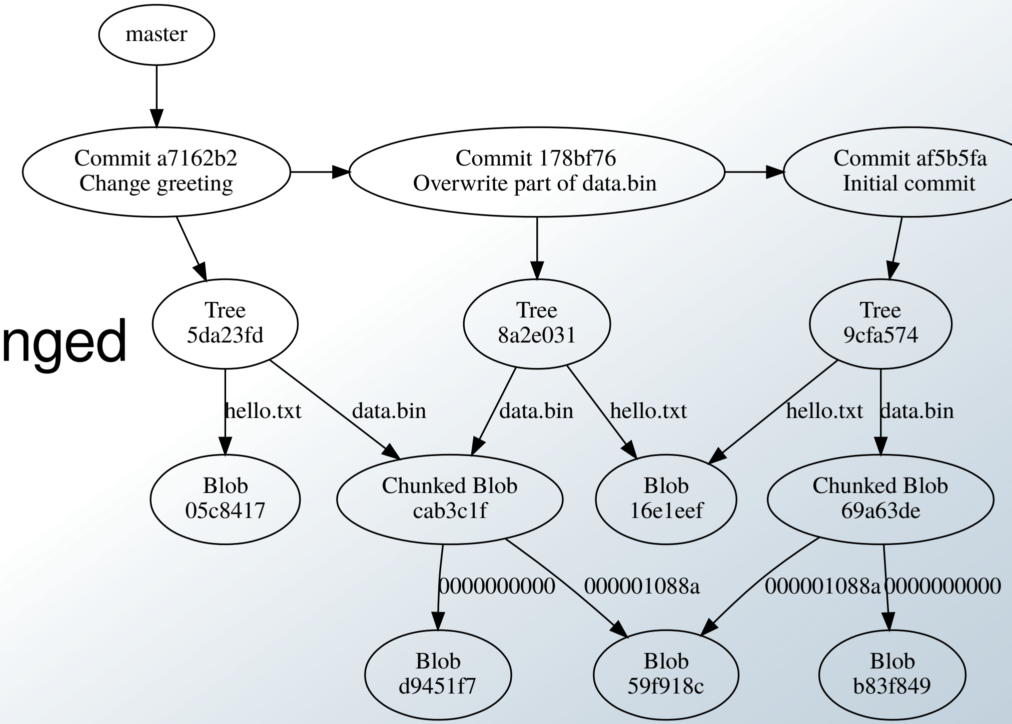
Directed Acyclic Graph (DAG)

- Content-addressable
  - Hash file → Blob
  - Hash dir → Tree
  - Hash state → Commit
- De-duplicates
- Immutable
- Append only
- Can always append
- Easy to sync
- Can verify hashes



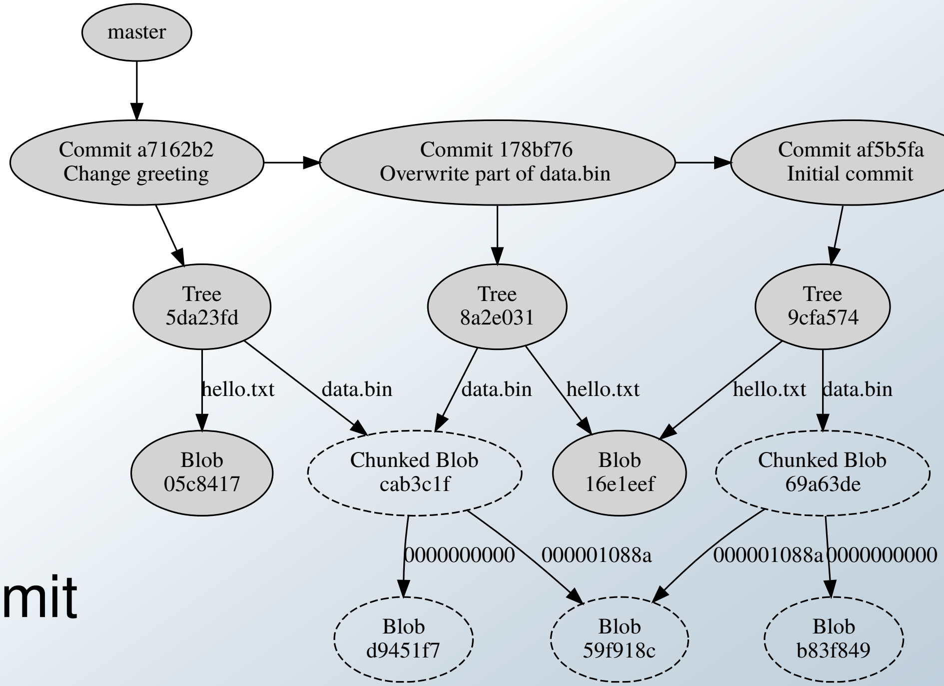
Handle Binary Files: Break files into chunks

- Use rolling hash
  ala rsync
- Binary files OK
- Chunks by content
- De-duplicates unchanged chunks
- Hash chunks
  → Chunked Blob



Prototype System: Distributed Media Versioning (DMV)

- Goal:
  - Handle binary/large files
  - Distribute repository
- Incomplete
- Basic version control features working
  - Commit / checkout
  - Log
  - Branch and merge
- Subset checkout/commit working



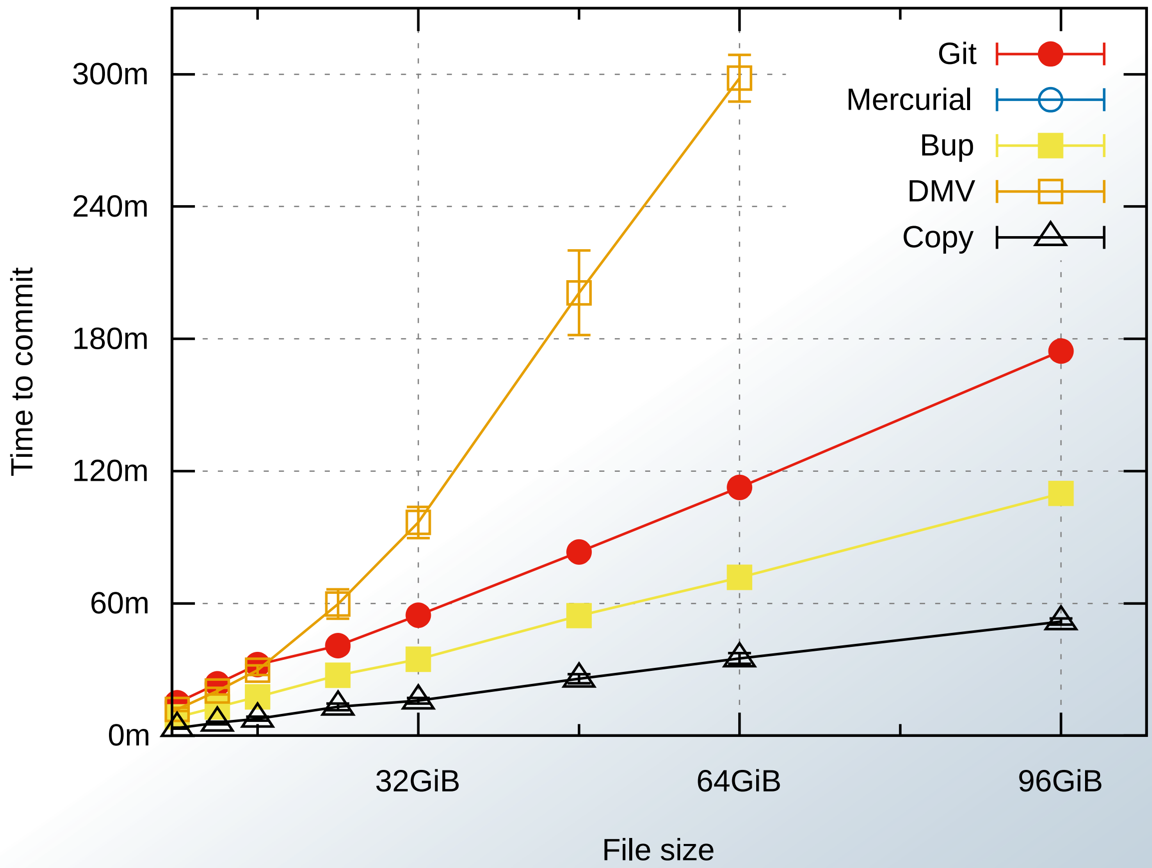
Experiment Methodology

- Four VCSs: Git, Mercurial, Bup, DMV prototype
- Control VCS: simple copy (`cp`)
- Python script:
  - Reformat partition
  - Generate random data (`/dev/urandom`)
  - Commit with VCS, record command run time (wall-clock)
- Data increases exponentially, includes intermediate steps:
  - File sizes 1 B to 96 GiB, by power of 2, w/ extra step at 1.5x:
    - 1 MiB, 1.5, 2, 3, 4, 6, 8, 12, 16, 24, 32, 48 ,64 …
  - File counts 1 to 10 M, by power of 10, w/ extra steps at 2.5x, 5x, 7.5x:
    - 10, 25, 50, 75, 100, 250, 500, 750, 1 000, …
    - Each file, 1 KiB

Experiment Platform

- 4 nearly identical PCs
  - One full experiment run on each
- Hardware
  - Hewlet Packard desktop PC
  - Intel Core 2 Duo CPU E8500 @ 3.16GHz
  - 8 GiB RAM
  - SATA hard disk (not SSD)
- Operating system
  - Debian Linux 8.6 (Jessie)
  - Linux Kernel 3.16.0
- Filesystem
  - 197 GiB LVM partition
  - Ext4 filesystem
  - 4 KiB block size
  - I/O scheduler: CFQ
- DMV compilation
  - Rust stable 1.15 or 1.16
  - `--release` compiler flag

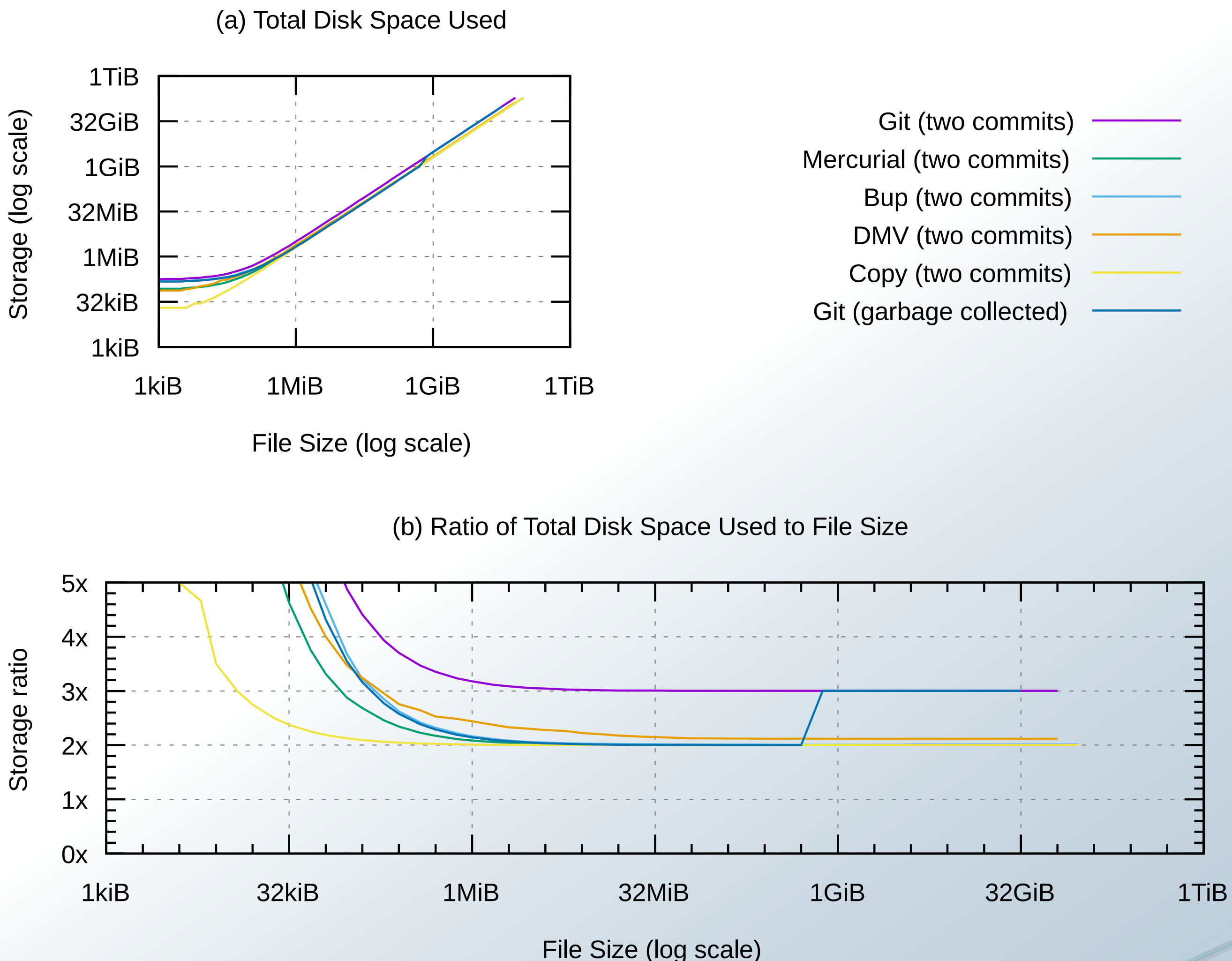
Results: Increasing File Size



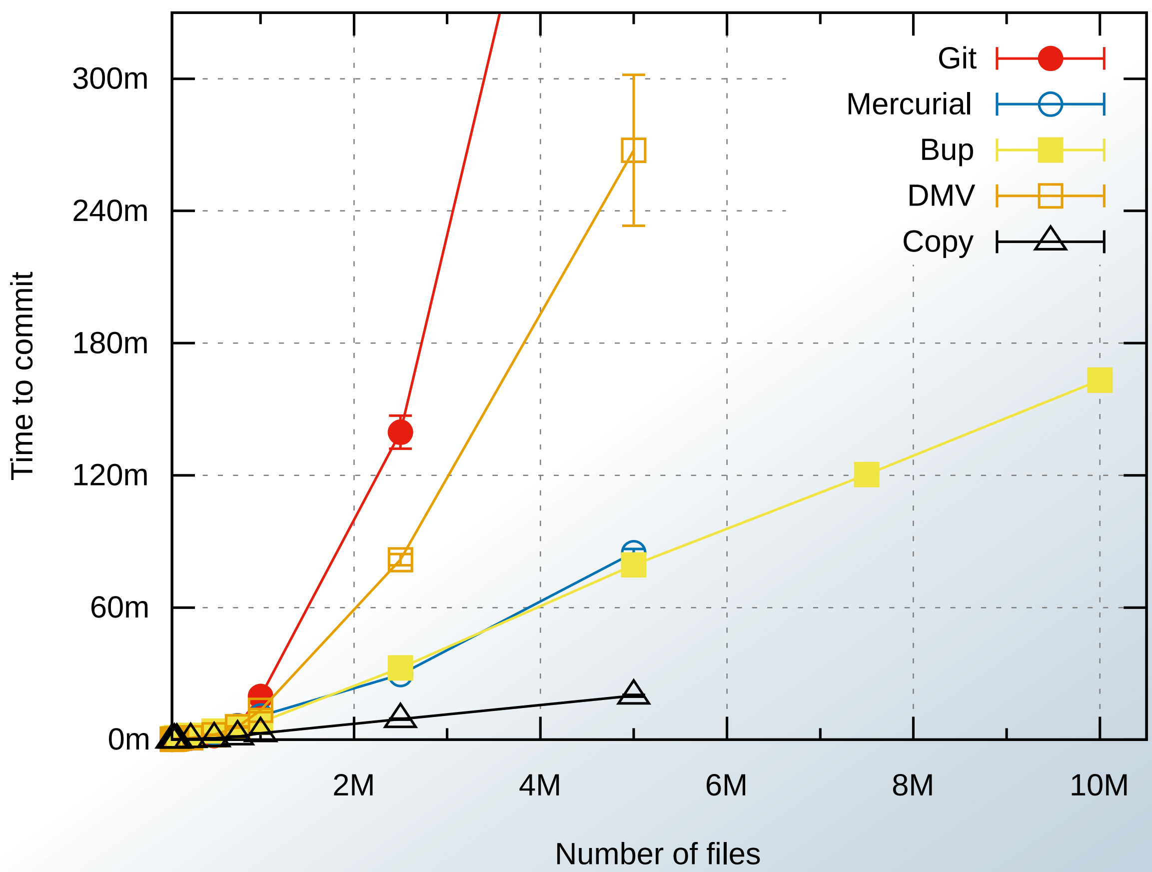
Results: Increasing File Size

- 96 GiB: largest that could fit 2x on 197 GiB partition
- 2 GiB
  - RAM limitations of diff algorithm come into play
  - Mercurial commit aborts with error message
  - Git commit succeeds, but gives warning, GC fails
  - DMV, Bup OK — break into chunks
- Speed
  - Copy fastest, then Bup, then Git. DMV slowest.
  - DMV converts “big file” problem into “many files” problem

Results: Increasing File Size — Repo Size



Results: Increasing Number of Files



Results: Increasing Number of Files

- Limitation: inodes
  - Store file-per-file: run out of inodes
  - 197 GiB partition: ~13 M inodes
  - 7.5 M files x2 $\rightarrow$ 15 M inodes $\rightarrow$ disk “full”
  - Bup OK — packs files together
- Speed: sequential vs random writes
  - Git and DMV slowest
    - filenames by hash, effectively random
  - Copy, Bup, and Mercurial fastest
    - Copy, Mercurial: filenames by input file names
    - Bup: append to pack files
    - Both: sequential writes

Conclusions

- We have rediscovered the limits of the Unix filesystem
  - 4 KiB block size: smaller files waste disk space
  - Total number of files limited by inodes
  - Filesystem optimized for sequential writes
  - Random file-name writes much slower
- Key to storing large files (>RAM): break into many files
- Key to storing many files (>inodes): pack back together
- In the process, restructure data as DAG
  - Content-addressable storage de-duplicates
  - Append-only structure easy to sync
  - Useful ways to shard data