AFS-Fuse Client-Server

Sam Kottler

Preeti Nayak

Asad Khan

POSIX Compliance

- We support open(), close(), creat(), unlink(), mkdir(), rmdir(), read(), write(), pread(), pwrite(), stat(), fsync(), and truncate()
- To match POSIX, writes are not persistent until fsync
- Unlike POSIX, writes are persistent after close
- creat(), unlink(), mkdir(), and rmdir() are persistent after returning because the changes must be on the server

Structure

- We keep a per-file attribute file on the client containing a dirty flag and the server's mtime.
- On open, if the file is in the cache, and the cached mtime matches the actual server mtime, there is no need to Fetch.
- On Fetch, get file from server, write the file to cache, then update attribute file.
- On Store, send file to server, then update attribute file. On the server, Store is last-close-wins.

Crash Recovery

Client

- Separate attribute file persisted after every update
- First open after crash, can continue to use same file if no updates on server
- Retry on server crash

Server

- No volatile state
- Atomic rename for Store

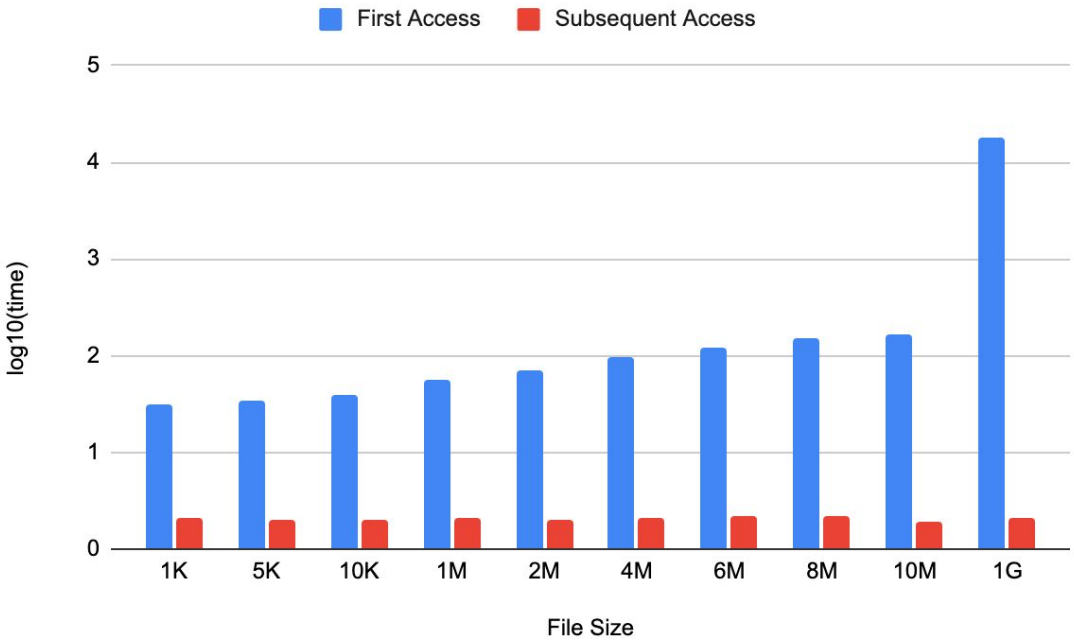
Protocol Diagrams

Open->Write->Close
FetchRPC(foo.file)
write(foo.file)
fsync(foo.file)
[write(foo.attr)]
fsync(foo.attr)
[write(foo.attr)]
fsync(foo.attr) xN
pwrite(foo.file)
fsync(foo.attr)
StoreRPC(foo.file)
[write(foo.attr)]

StoreRPC
open(foo.cli_ID)
write(foo.cli_ID) xN
fsync(foo.cli_ID)
[rename(foo.cli_ID,foo.file]

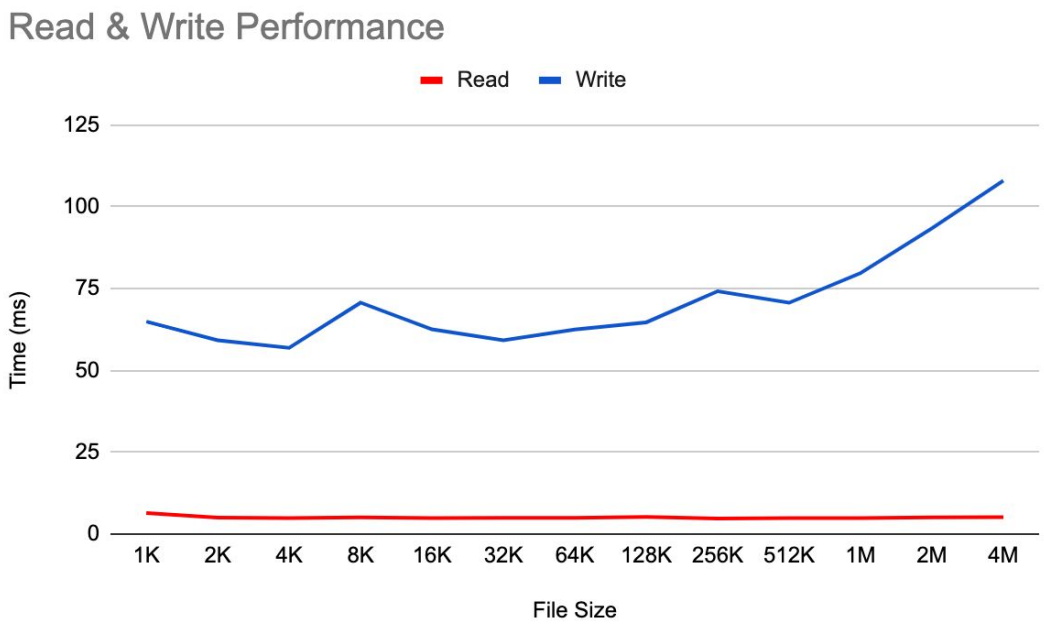
Demos!

Performance



Due to caching the subsequent accesses take less time

Performance



Optimizations

- If the open() flag is set to O_TRUNC, avoid fetching from the server
- Caching the attr files in the memory would help if the same file is opened again and again
- Using threadpool at the server to handle multiple clients concurrently
- Multiple end users at a client machine can be supported by writing the changes to a temp file (one for each user) and renaming the file on close()
- Implement a reply-cache on the server for non-idempotent operations.