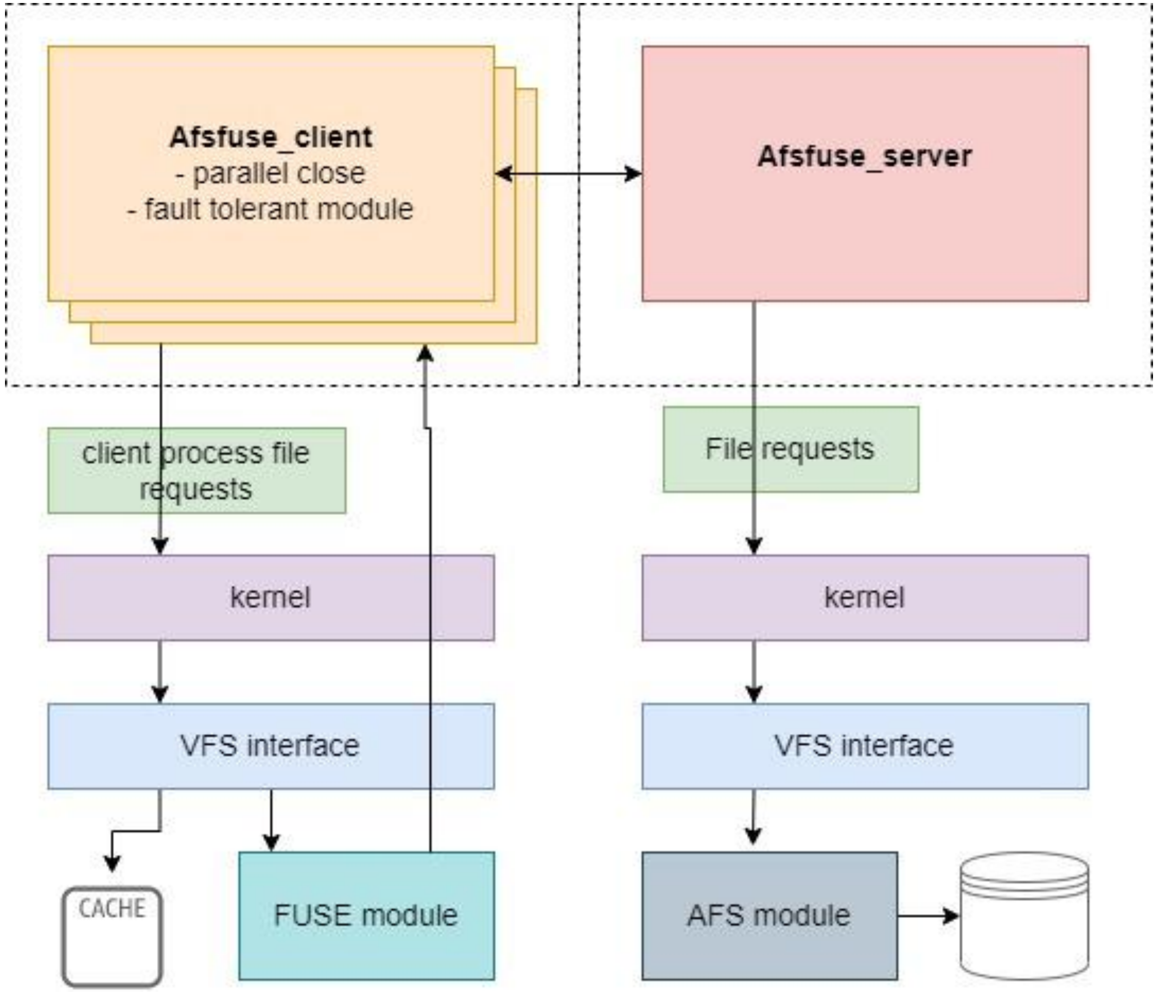
AFS (like) Distributed File System

CS 739, Project - 2

Divyanshu Kumar
Mohit Loganathan
Salman Munaf
(Dept. of Computer Sciences/CDIS)

System Design



- **No callbacks**! Workaround for avoiding stale cache - stat for checking modified time and fetch if changed. Flush file to server on close only if it’s changed.

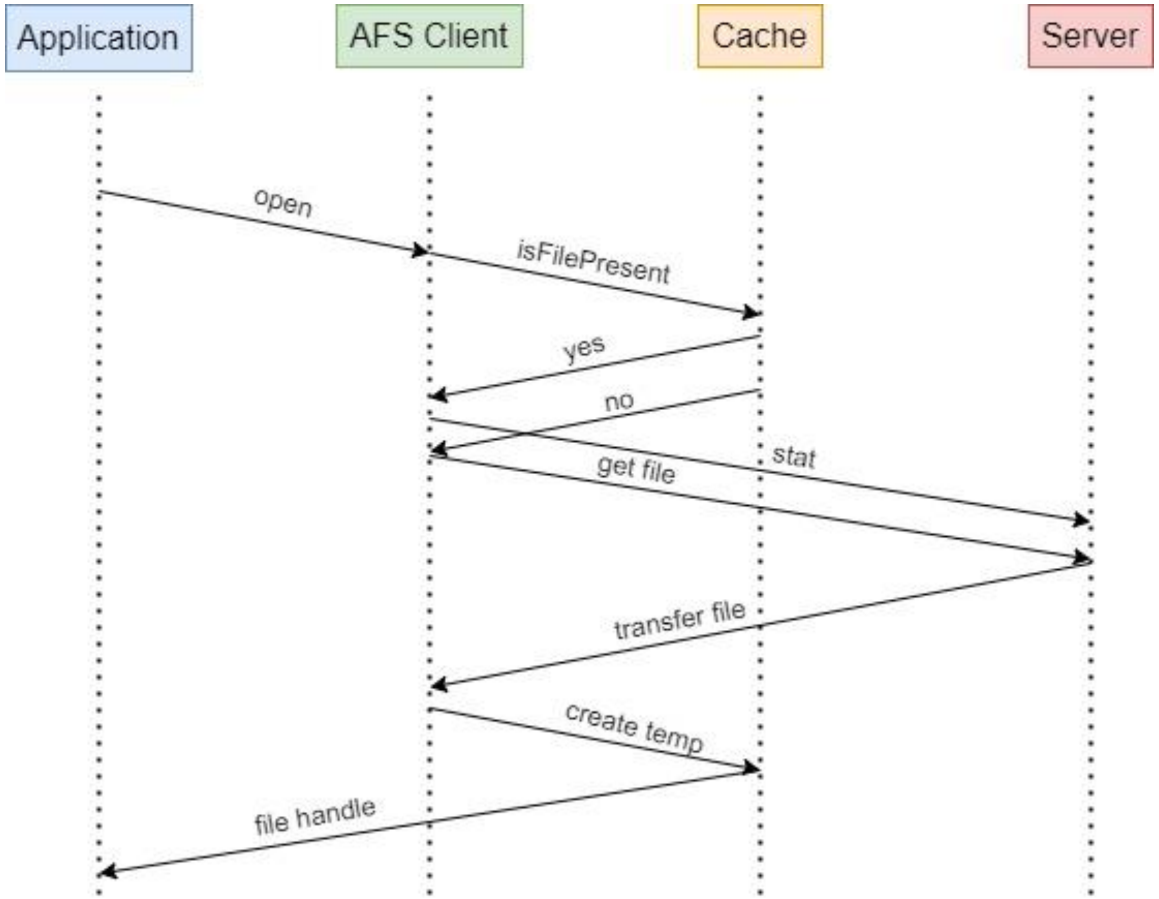
- **Multiple Users** - Last writer wins (rather last renamer wins, more on that later..). Temporary file copy for giving concurrent access of the same file to multiple applications

- **Client Crash** - Recovery protocol to check unfinished close( ) calls.

- **Server Crash** - Client distinguishes between gRPC fail and operation fail. If gRPC fails, client retries with exponential backoff (limited to 10s)

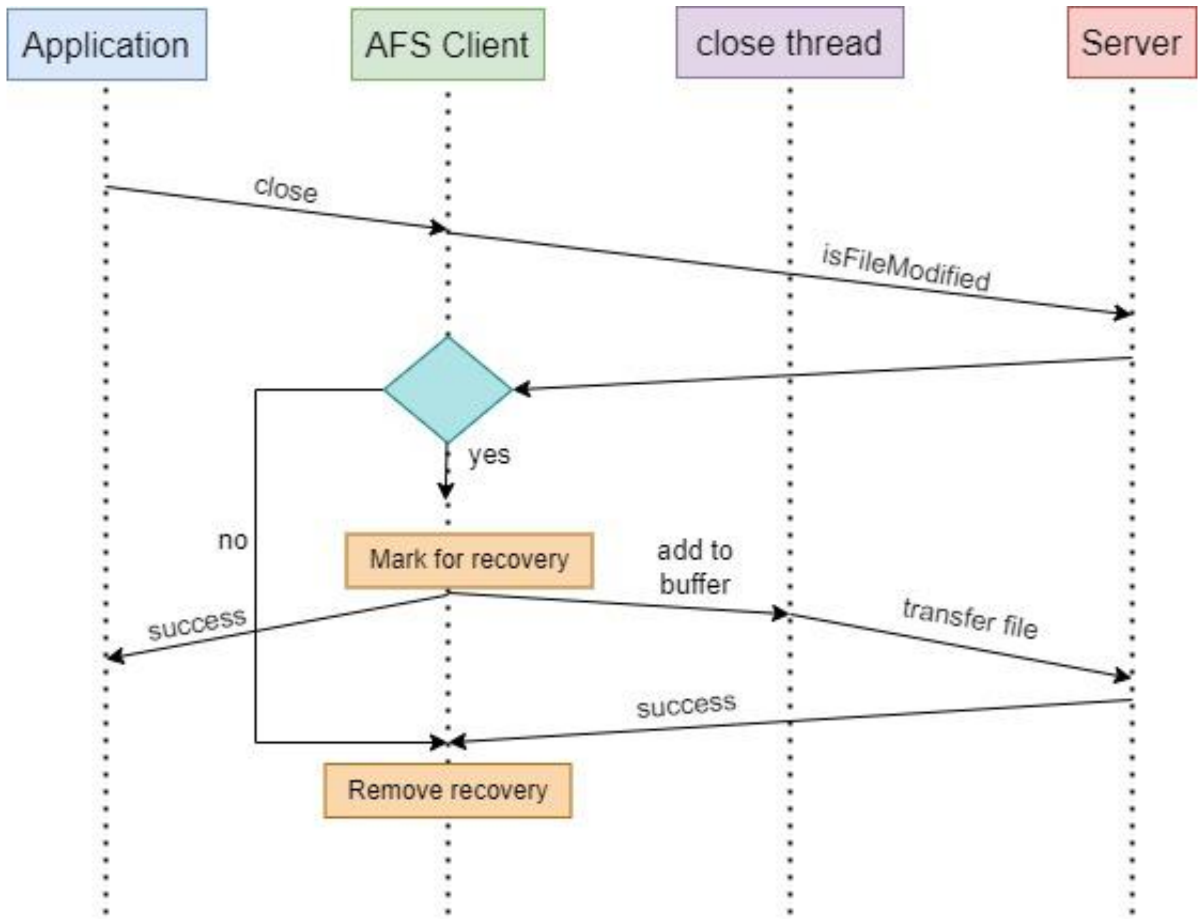
Open()

- Key design choices:
  - Only fetch file if not present locally or if it is modified at server.
  - Create a temporary file copy for every open so that applications can parallely write to same files without overwriting each other.
  - Create (Special case of open) - In create call, a create RPC on server is also executed.



Close()

- Key design choices:
  - Stat file at server to see if local file is modified at a later timestamp than server.
  - Rename file to a special ‘recover’ file so that if client crashes before saving file at server, it can recover at reboot.
  - Use a thread to parallely send file to server. At completion, the thread un-marks the file for recovery.
  - Close thread - A shared queue with main process where close() can submit request to send files to server.
  - Close thread - Improves perceived close time to application.



Supported File System Operations

Local Calls

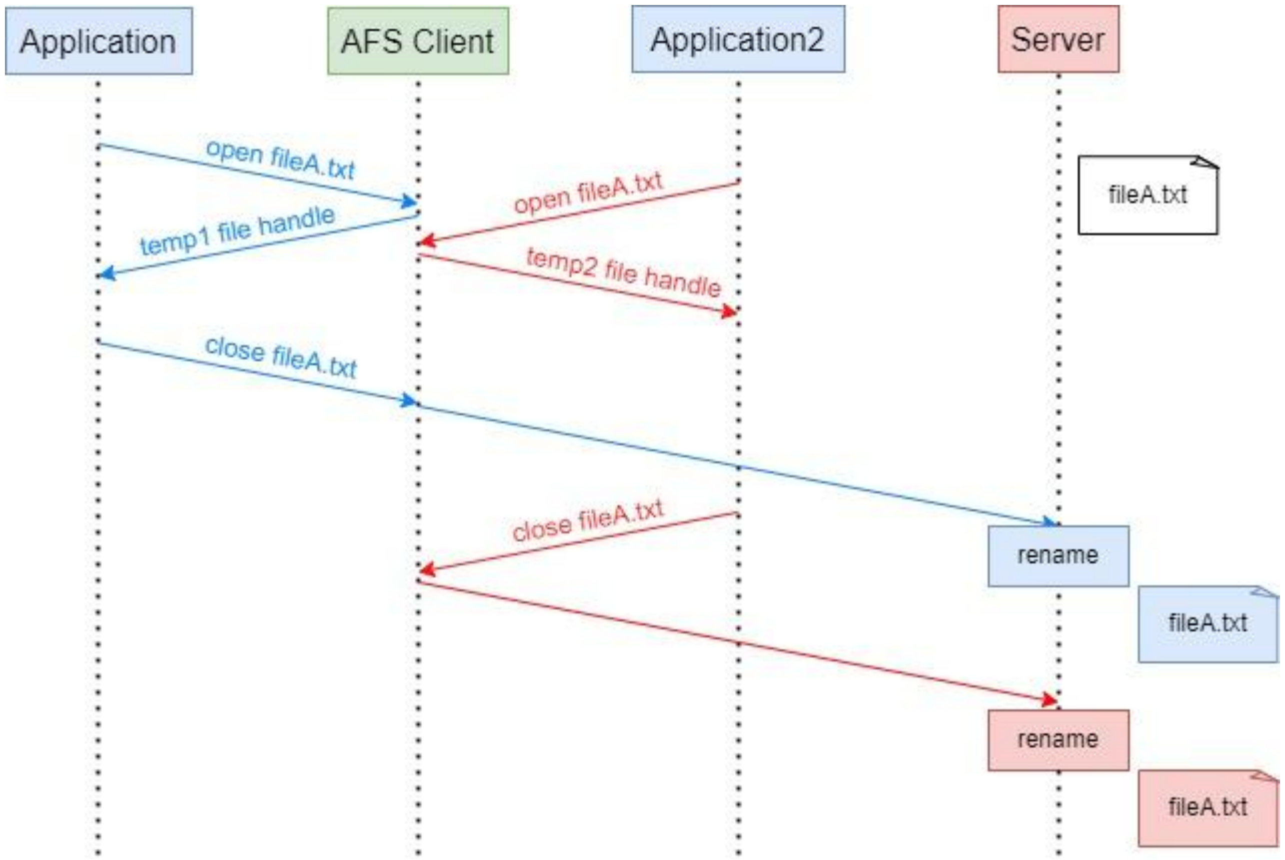
read( )
write( )
flush( )
fsync( )

Remote Calls
(at least one RPC)

getattr( )
readdir( )
open( )
create( )
mkdir( )
rmdir( )
unlink( )
rename( )
utimens( )
release( )

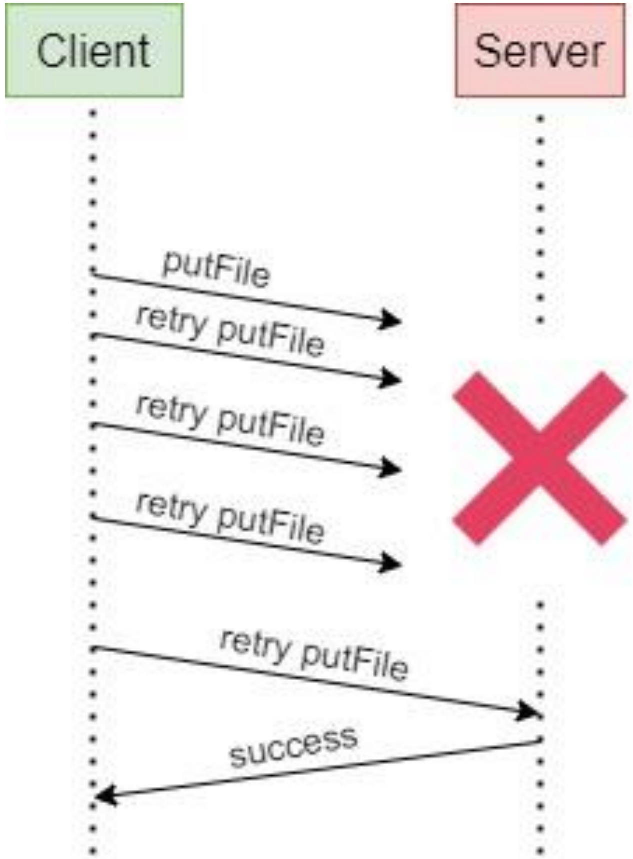
Last writer wins

- Key design choices:
  - Create temporary file to receive data at server and rename to make the receive of file atomic.
  - *<u>'Last Renamer wins'</u>*
  - Similarly at client, last application to close the file wins.



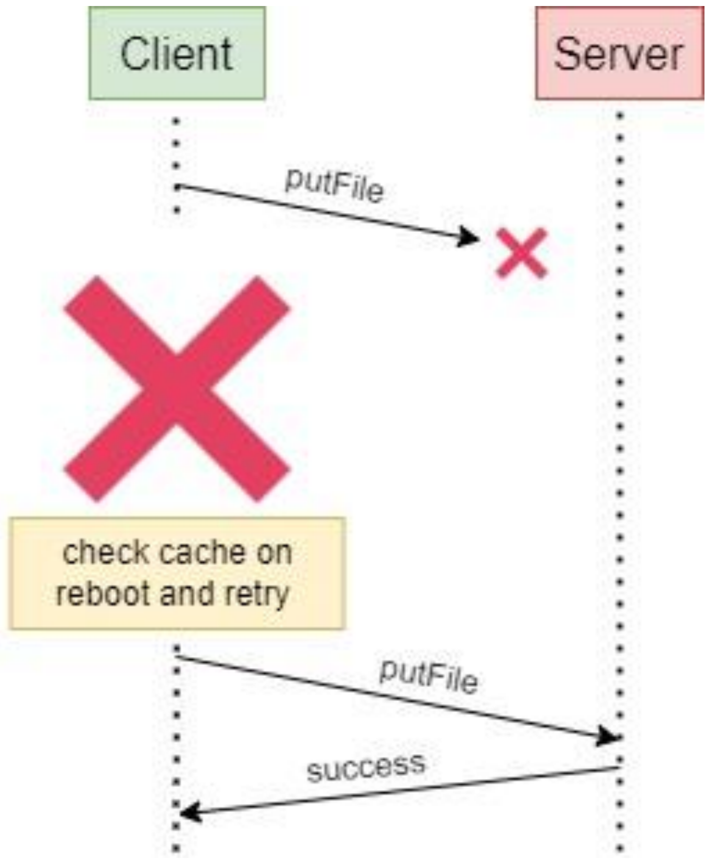
Server crash

- Key design choice:
    - The client times out at gRPC and retries the request with exponential backoff limited upto 10 seconds.
    - If operation is streaming, then at server reboot whole operation is restarted and previous chunks are discarded.



Client crash

- Key design choice:
    - We support recovery of files on which a close( ) call was invoked.
    - Mark at file for recovery at close( ) and check the file system at init time if there is any recovery file.
    - If a recovery file is present, modification time at client and server is compared and file is only sent to server if client's version of the file is new.
    - No support of recovery of un-closed( ) files. Can be supported, but hard to distinguish between write finished files vs write unfinished files.

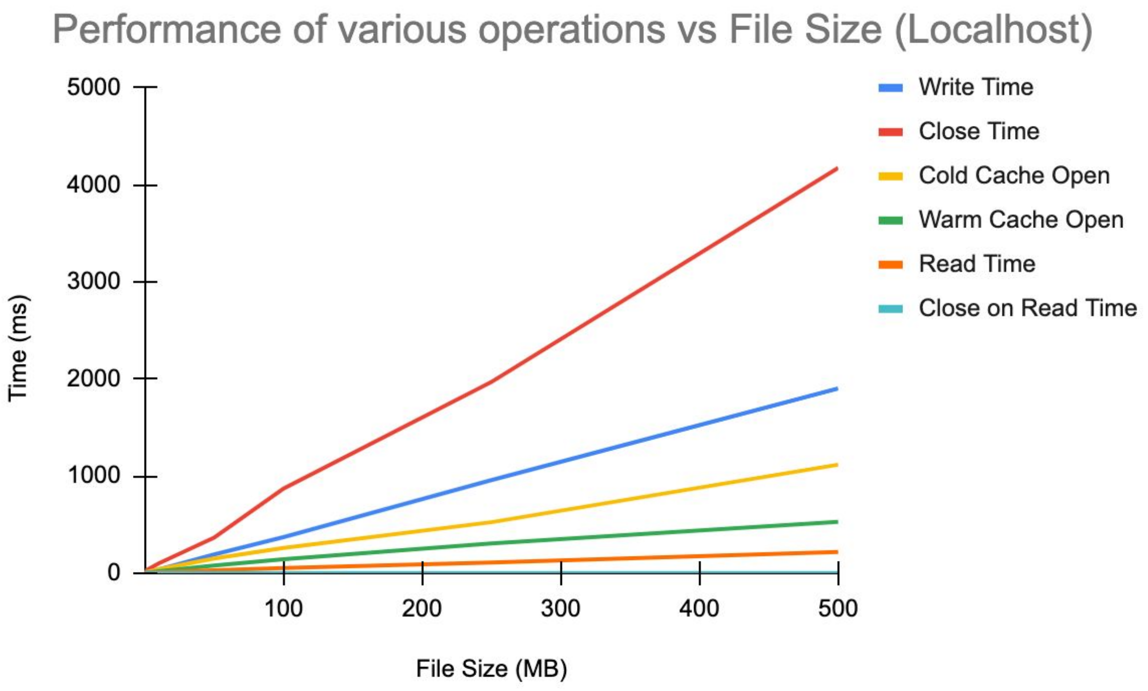


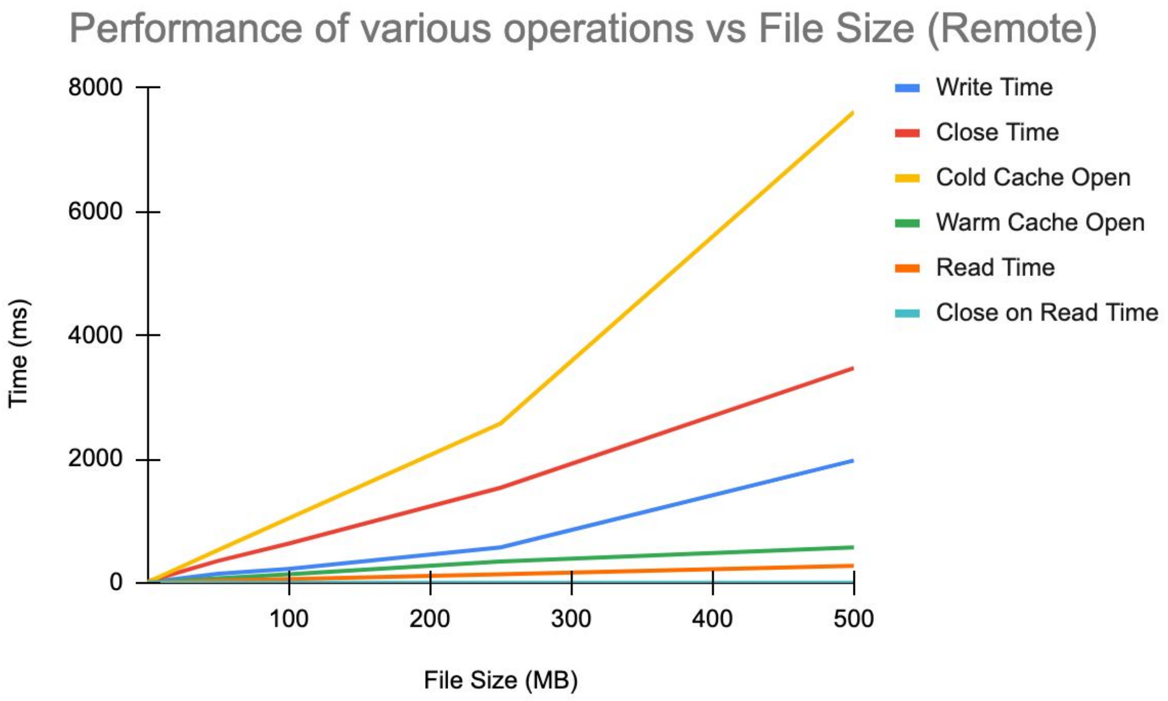
Performance - Our AFS vs Local FS(ext3)

	Local FS	AFS
Read (MB/s)	4055	1900
Write (MB/s)	1030	601

- Not too bad!
- Our observations-
  - Our AFS client is a user-level process so there is lot of re-routing of calls from kernel to AFS process and back.
  - This especially appears in write case where the major bottleneck is not process level communication but probably the write overhead.

Performance





LocalHost

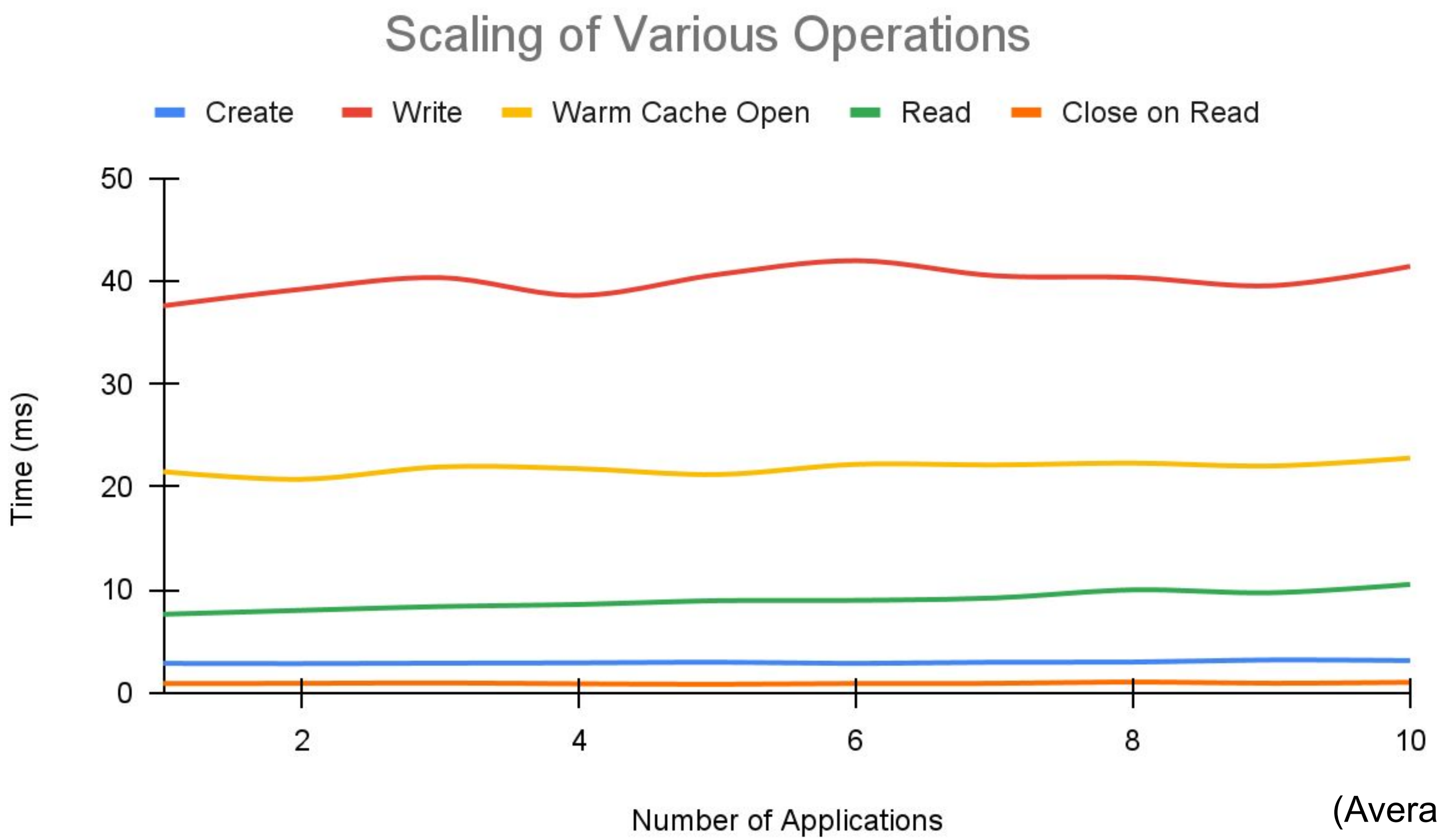
Localhost shows the performance without any network delay.

Remote

Remote client-server shows that cold cache open is affected the worst understandably so due to network delay.

Scalability - (1)

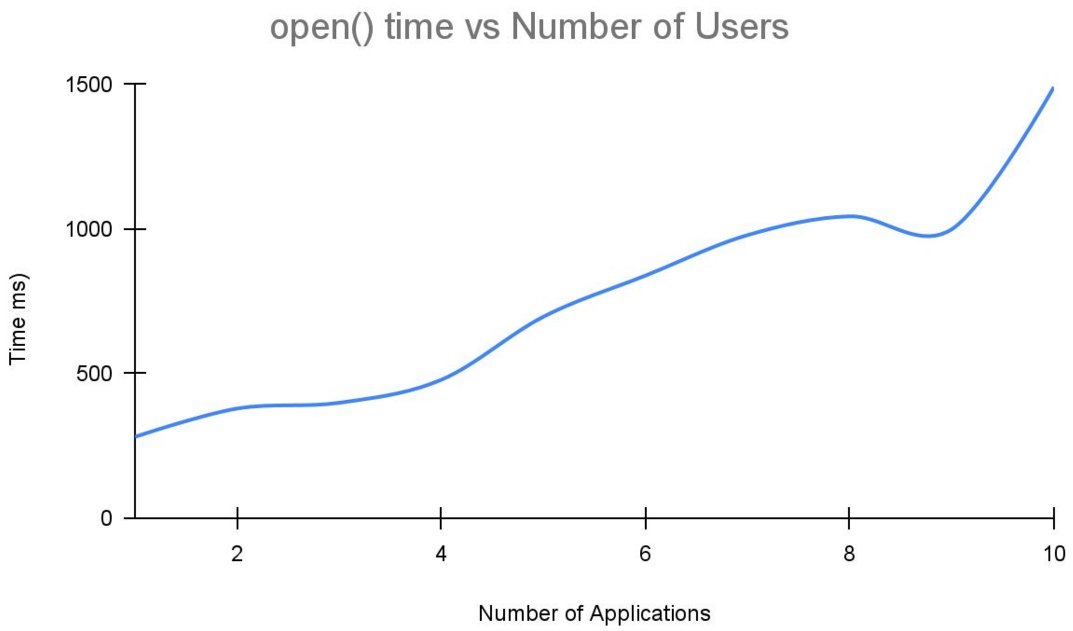
As expected local file system operations don't change much on scaling out.

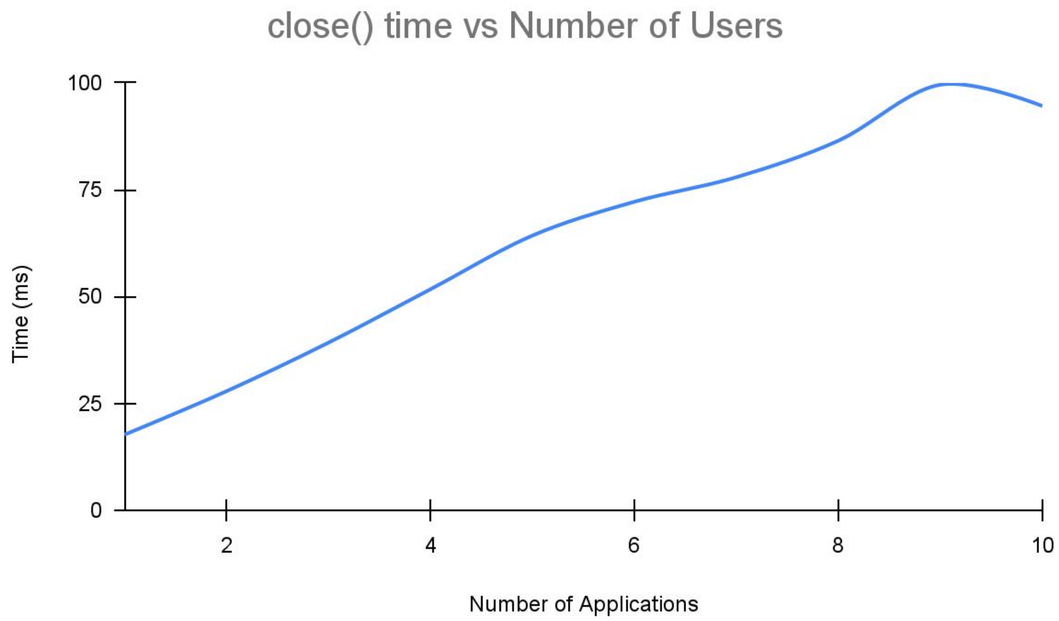


(Average time for a 10 MB file)

Scalability - (2)

As number of users increase, the performance of the operations that does depend on server interaction also degrade.





(Average time for a 10 MB file)

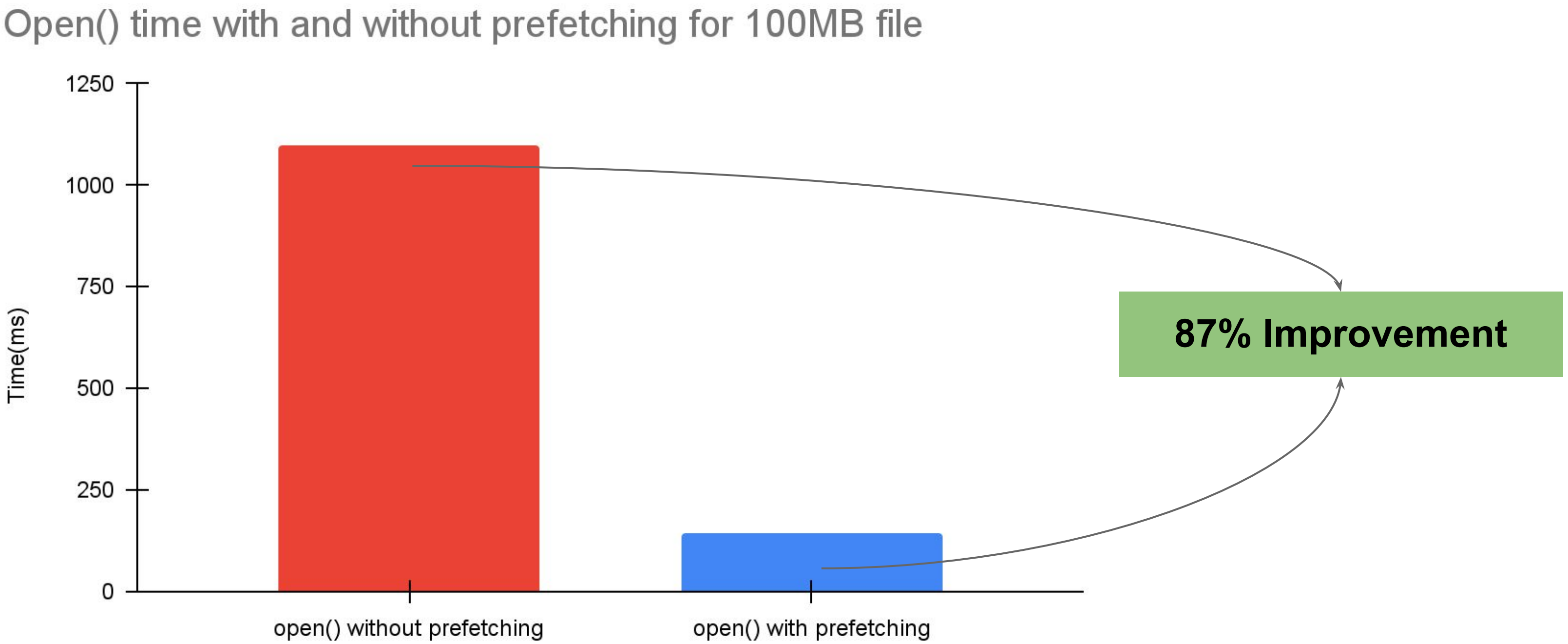
Improvements

- **Eager fetching** of files in current directory
- **Streaming files** in chunks of 1 MB between clients and server (as opposed to sending complete file as one buffer - Not Scalable!)
- Only **send modified** files to server.
- Don’t fetch **unmodified cached files** again on open.
- Parallel sending of file to server - Making **Close ( ) non-blocking on send.**
- **Fsync data randomly** on Write ( ) - improves Close ( ) time.
- Making **temporary files** for writing to **avoid intermixed writes.**
- **Recovery of files** intended to be saved (called close on) by application.

Improving Open Time

Problem - First open time is huge
Cause - Transferring file to client over network takes time.
Idea - *Prefetch files of size upto a threshold in the current directory.*

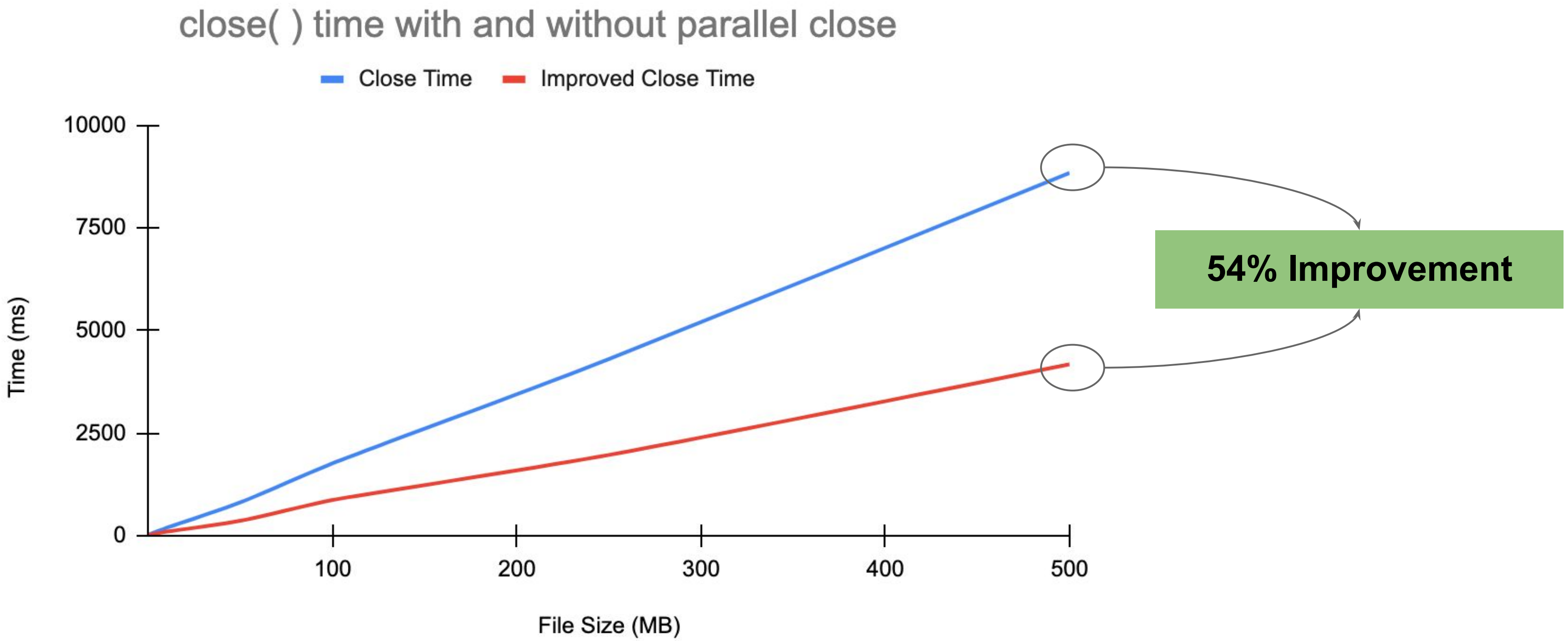
Observations:



Improving Close Time - (1)

Problem - Close time for big files is large
Cause (1) - Replicating file on server over network takes too much time.
Idea (1) - *Parallely send the file to server and return to user asap.*

Observations:

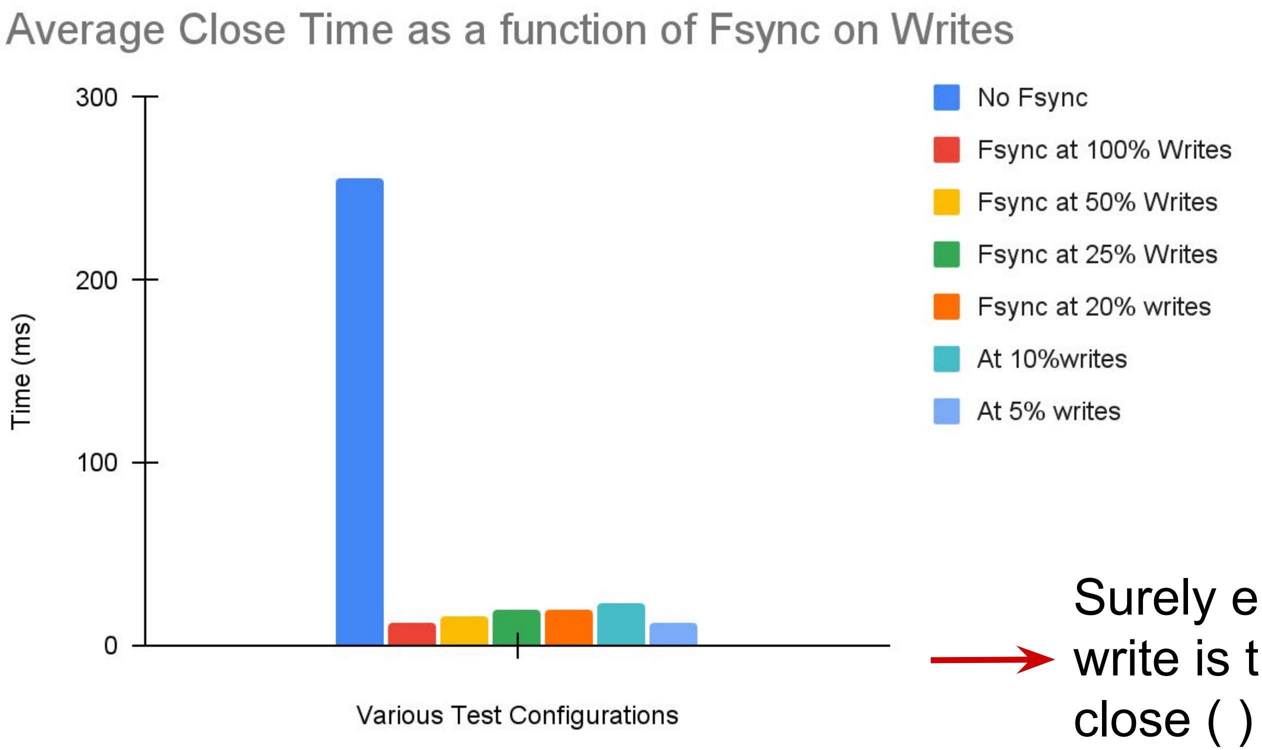


54% Improvement

Improving Close Time - (2)

Problem - Close time for some big files is large
Cause - Write buffering cause huge flush at Close time
Idea - *Spread fsync of write buffered data at Write Time*

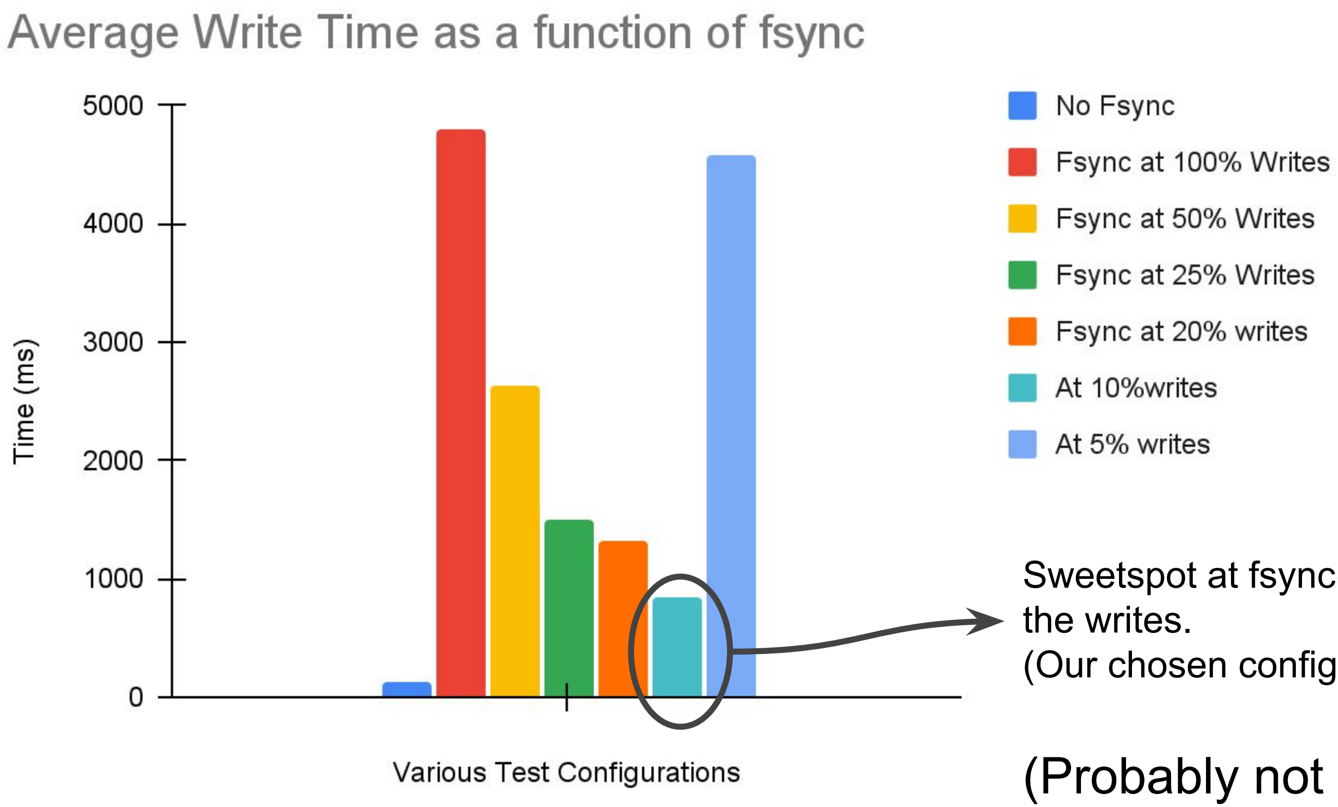
Observations:



Surely enough fsync at every write is the best outcome for a close ( ) call but..

Improving Close Time - (2)

Cons - Degrades write throughput

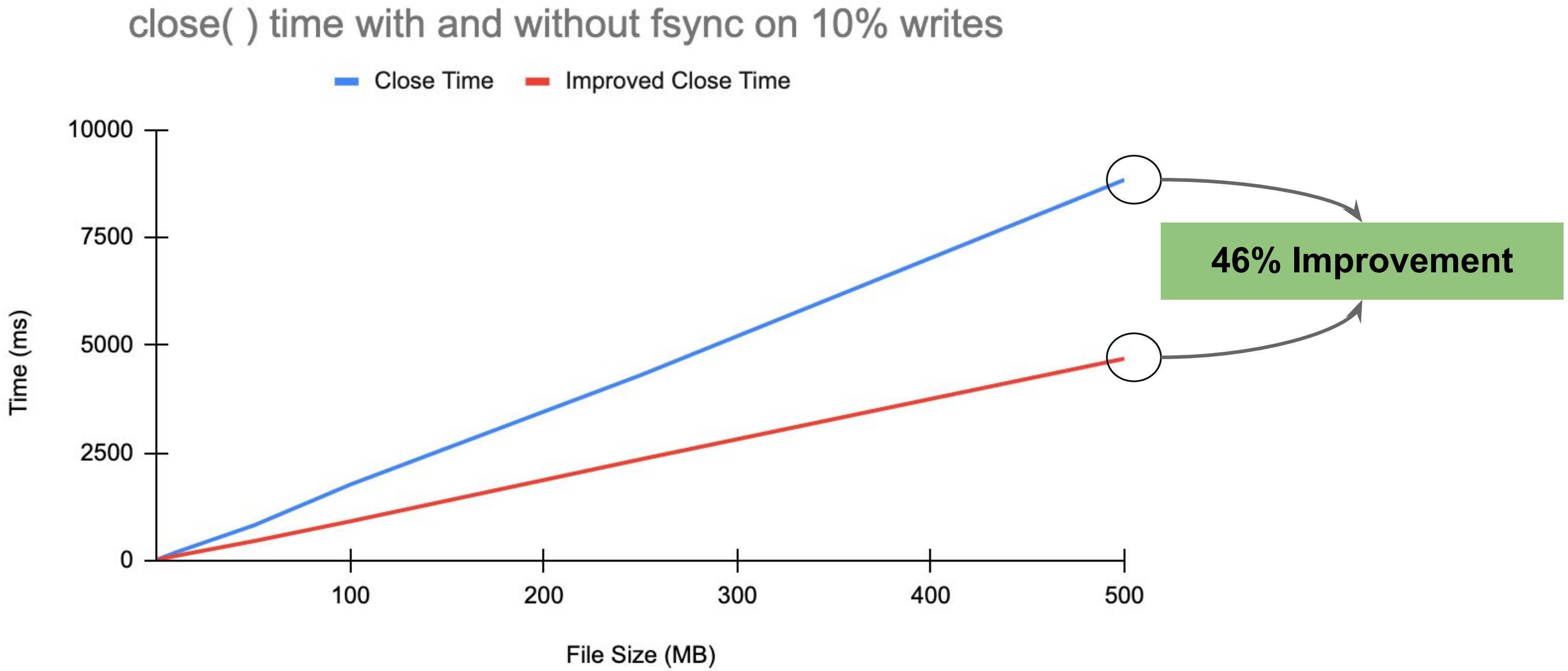


Sweetspot at fsync() at 10% of the writes.
(Our chosen configuration)

(Probably not bad for normal applications whose writes are anyways interspersed)

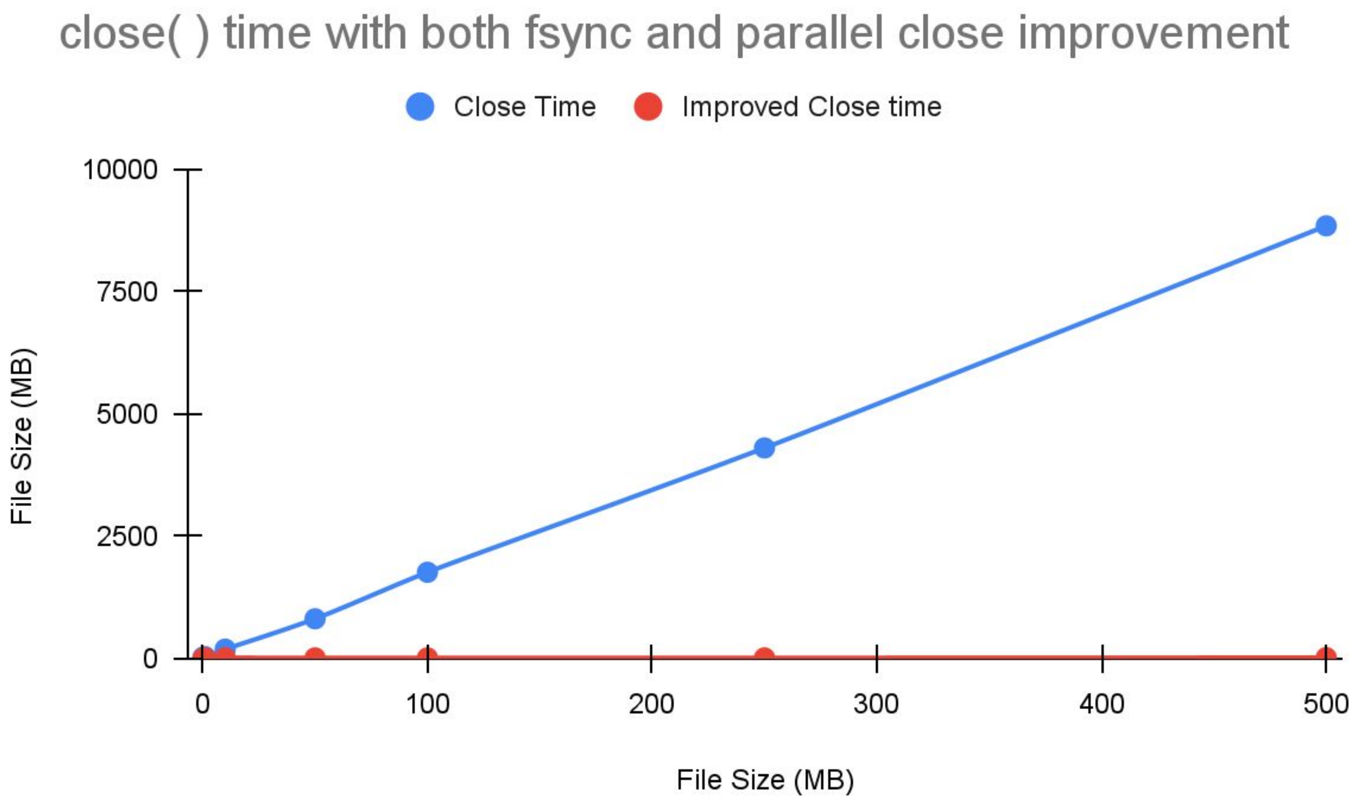
Improving Close Time - (2)

fsync at 10% of the writes:

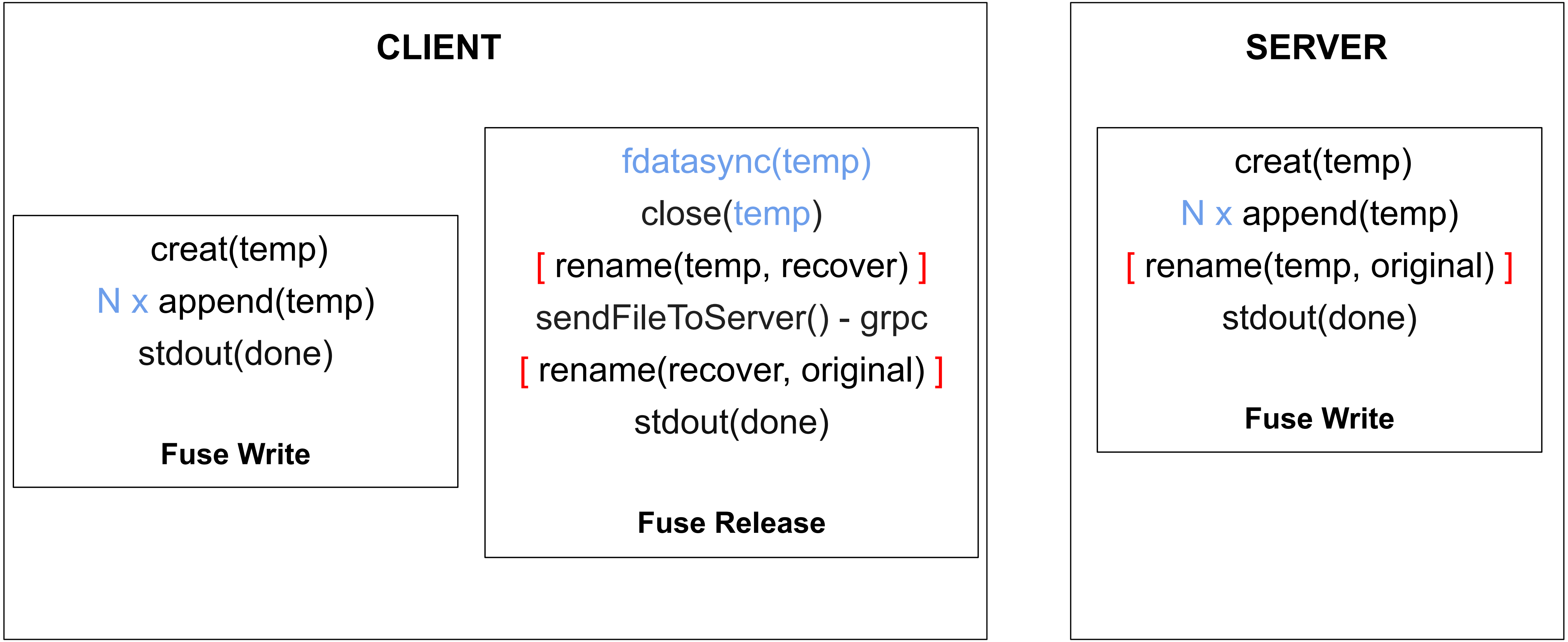


Improving Close Time - (1) & (2)

***From 8.8 sec to 21 ms…** (Also independent of file size now)*



Consistency protocol



CLIENT

creat(temp)
N x append(temp)
stdout(done)

Fuse Write

fdatasync(temp)
close(temp)
[rename(temp, recover)]
sendFileToServer() - grpc
[rename(recover, original)]
stdout(done)

Fuse Release

SERVER

creat(temp)
N x append(temp)
[rename(temp, original)]
stdout(done)

Fuse Write

Consistency protocol

- Key design choices:
  - We are using a protocol similar to the ext2 file system to maintain consistency.
  - During boot, the client scans the cache and checks if there are recovery (.recover) files present.
  - Stat file at server to see if recovery files are modified at a later timestamp than server.
  - Transfer all those files to server

On to the fun part..
Demo!

Thank You!

Q/A?

WISCONSIN

UNIVERSITY OF WISCONSIN–MADISON