CS 739 P2 - AFS like Distributed FS

Group 4
-Lincoln Spartacus James
-Kaustubh Khare
-Prabhav Adhikari
-Madhav Kanbur

High Level Architecture

Client

- Whole file-caching, replicates server's directory structure.
- Maintains a mapping file on disk containing <filename, server's last modified>
- open() fetches file along with it's last modified.
- read()/write() is always local.
- close() flushes back to server (if dirty) and gets back server's new last modified time.

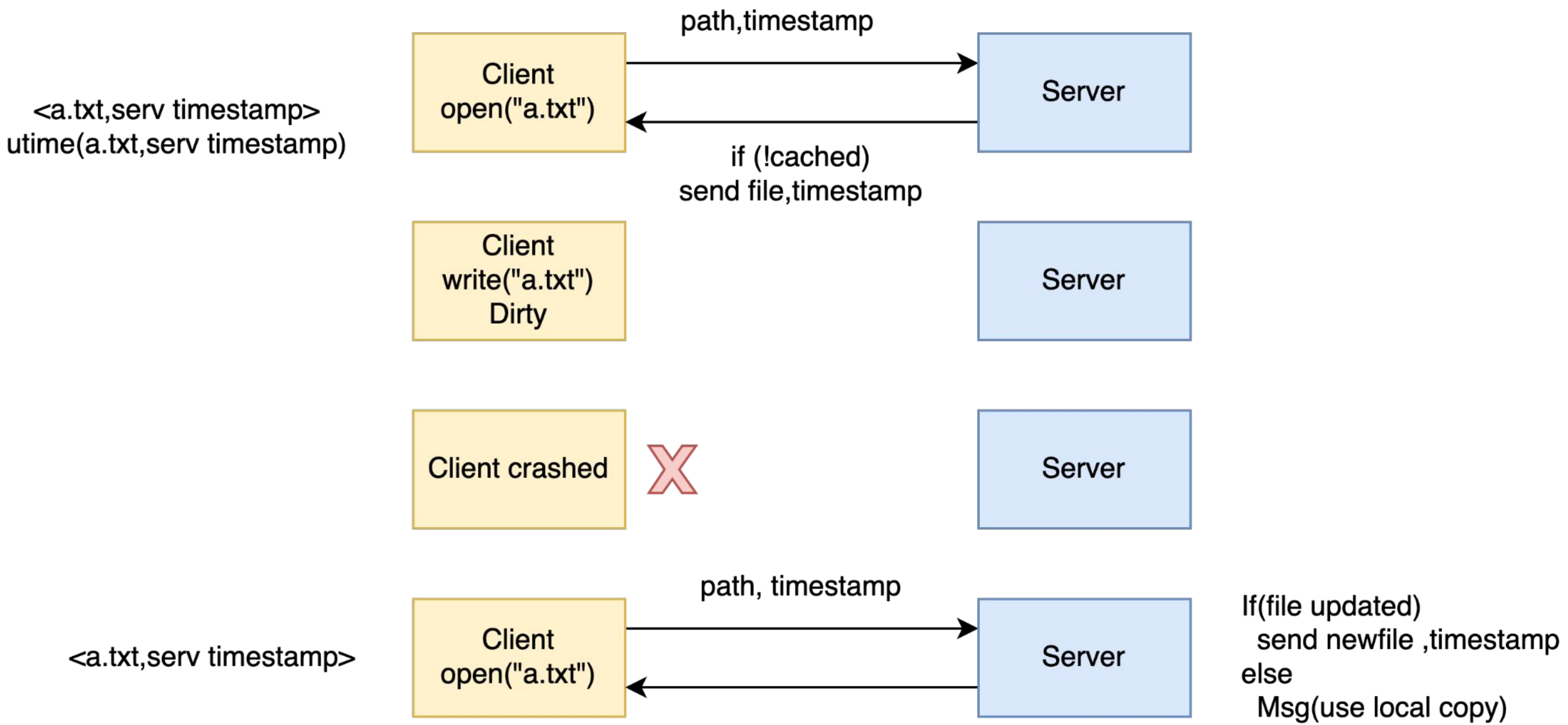
Server

- No state on server.
- Uses a local update protocol (write, rename) for atomicity.

AFS Protocol

- On every open(), Client sends its last modified from the mapping file (AFS_RECHECK). Server compares and sends back the new file (if updated) or an ALREADY_CACHED msg.
- close() flushes back to server (if dirty) and gets back server’s new last modified time.
- Last writer wins

Crash Recovery Protocol (Client)



Crash Recovery Protocol (Server)

… Do nothing. Reboot and continue as usual.

Server’s local error codes are sent back for all operations. This allows the client to distinguish between I/O errors on the server vs. Server crashes.

Durability (Server side)

- Since the server is stateless, no extra work required.
- Durability => file updates are atomic.
- Achieved with local update protocol (write to temp file, then rename)
- This ensures no mixing of data occurs.

Durability (Client side) : Case 1 - File creation

Underlying FS : ext3-ordered

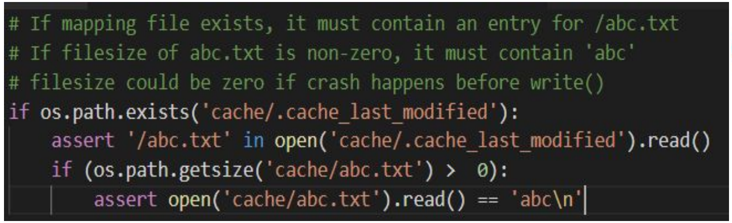
Operation :

fd = creat('abc.txt');

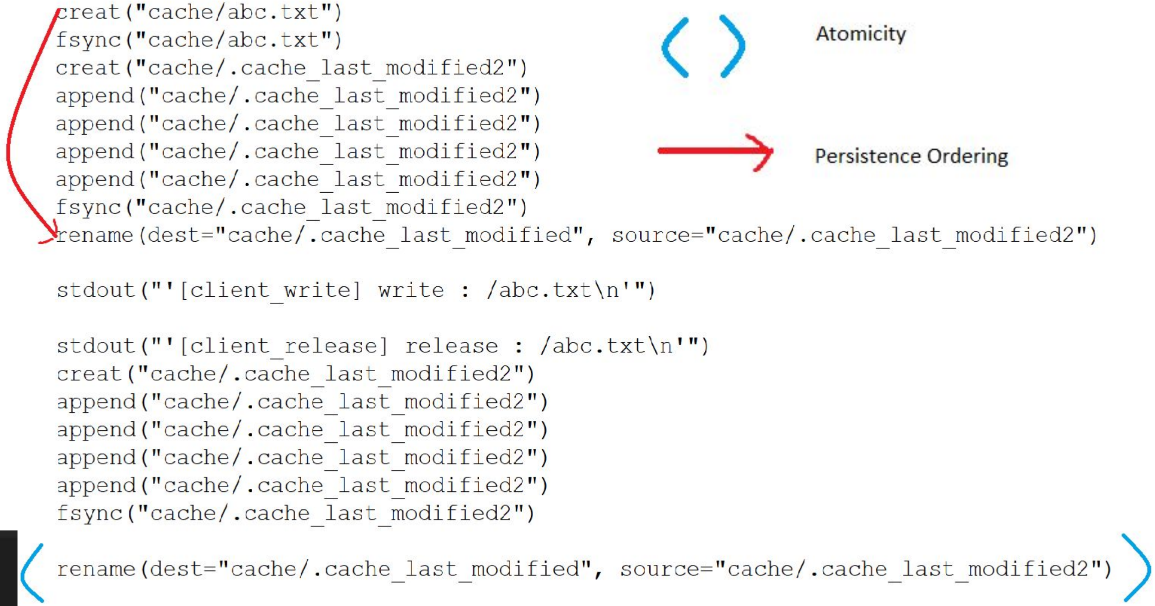
write(fd, 'abc\n', 4);

close(fd);

Checker code :



```
# If mapping file exists, it must contain an entry for /abc.txt
# If filesize of abc.txt is non-zero, it must contain 'abc'
# filesize could be zero if crash happens before write()
if os.path.exists('cache/.cache_last_modified'):
    assert '/abc.txt' in open('cache/.cache_last_modified').read()
    if (os.path.getsize('cache/abc.txt') > 0):
        assert open('cache/abc.txt').read() == 'abc\n'
```



```
creat("cache/abc.txt")
fsync("cache/abc.txt")
creat("cache/.cache_last_modified2")
append("cache/.cache_last_modified2")
append("cache/.cache_last_modified2")
append("cache/.cache_last_modified2")
append("cache/.cache_last_modified2")
fsync("cache/.cache_last_modified2")
rename(dest="cache/.cache_last_modified", source="cache/.cache_last_modified2")

stdout("'[client_write] write : /abc.txt\n'")

stdout("'[client_release] release : /abc.txt\n'")
creat("cache/.cache_last_modified2")
append("cache/.cache_last_modified2")
append("cache/.cache_last_modified2")
append("cache/.cache_last_modified2")
append("cache/.cache_last_modified2")
fsync("cache/.cache_last_modified2")

rename(dest="cache/.cache_last_modified", source="cache/.cache_last_modified2")
```

< > Atomicity

→ Persistence Ordering

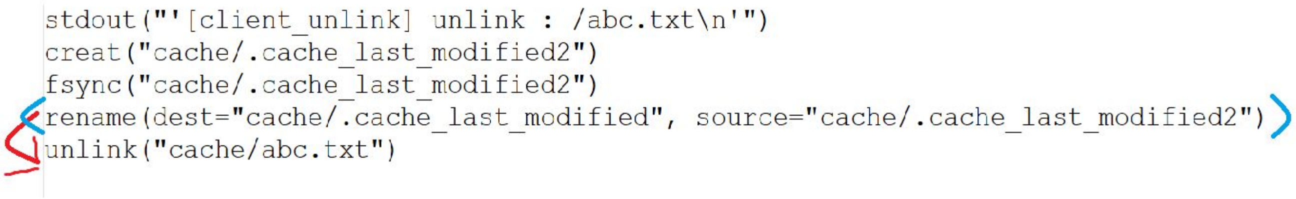
Durability (Client side) : Case 2 - File deletion

Underlying FS : ext3-ordered

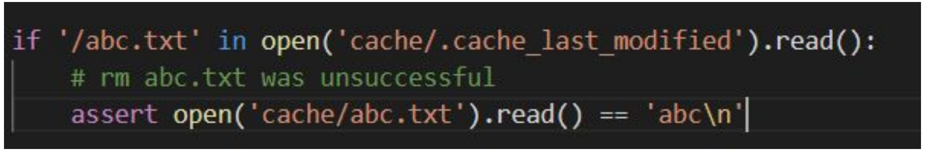
Operation :

rm /abc.txt

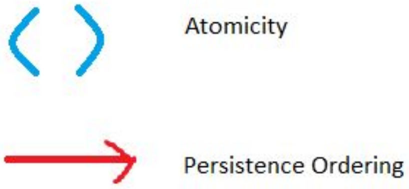
Checker Code :



```
stdout("'[client_unlink] unlink : /abc.txt\n'")
creat("cache/.cache_last_modified2")
fsync("cache/.cache_last_modified2")
rename(dest="cache/.cache_last_modified", source="cache/.cache_last_modified2")
unlink("cache/abc.txt")
```

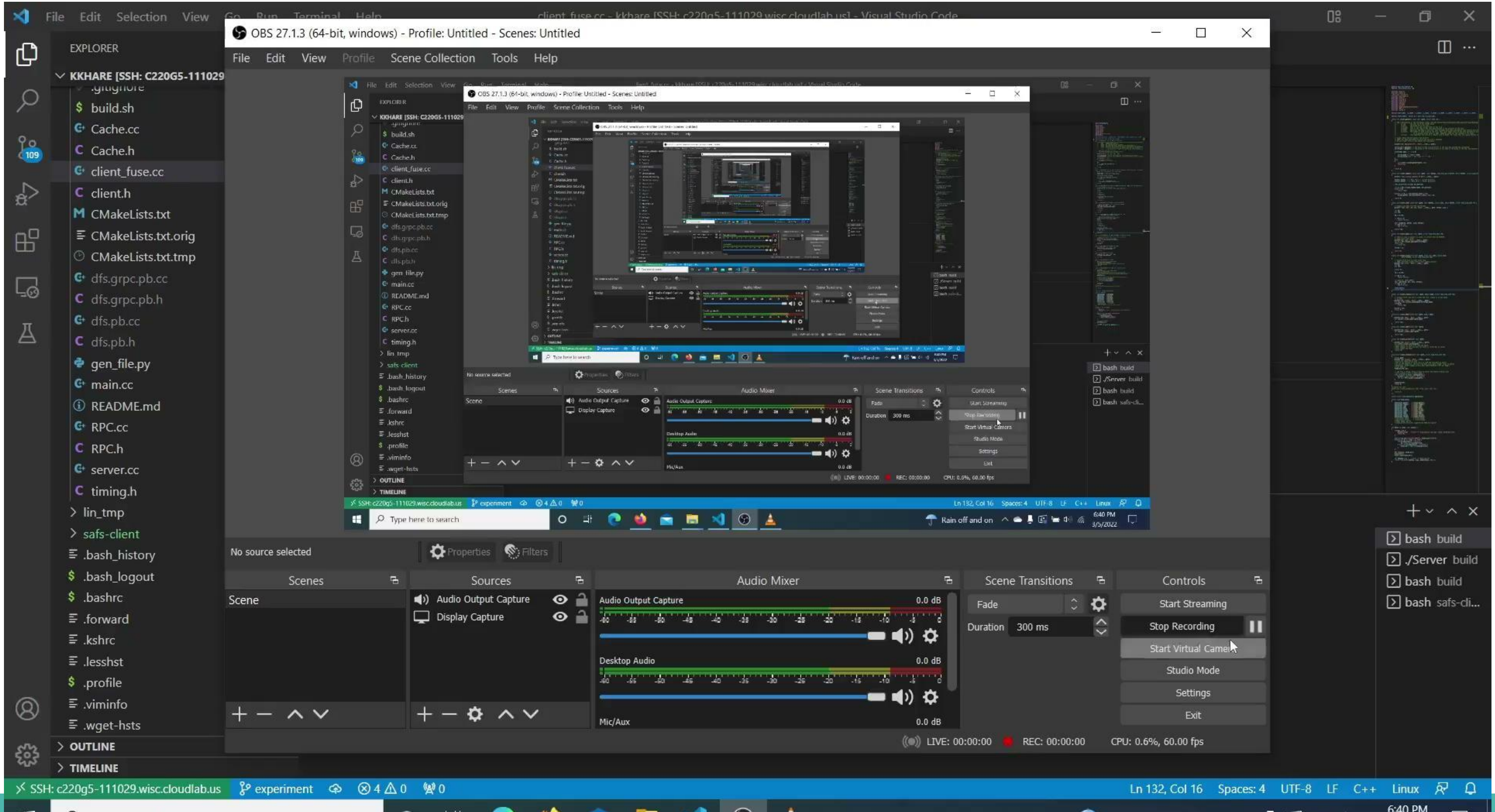


```python
if '/abc.txt' in open('cache/.cache_last_modified').read():
    # rm abc.txt was unsuccessful
    assert open('cache/abc.txt').read() == 'abc\n'
```



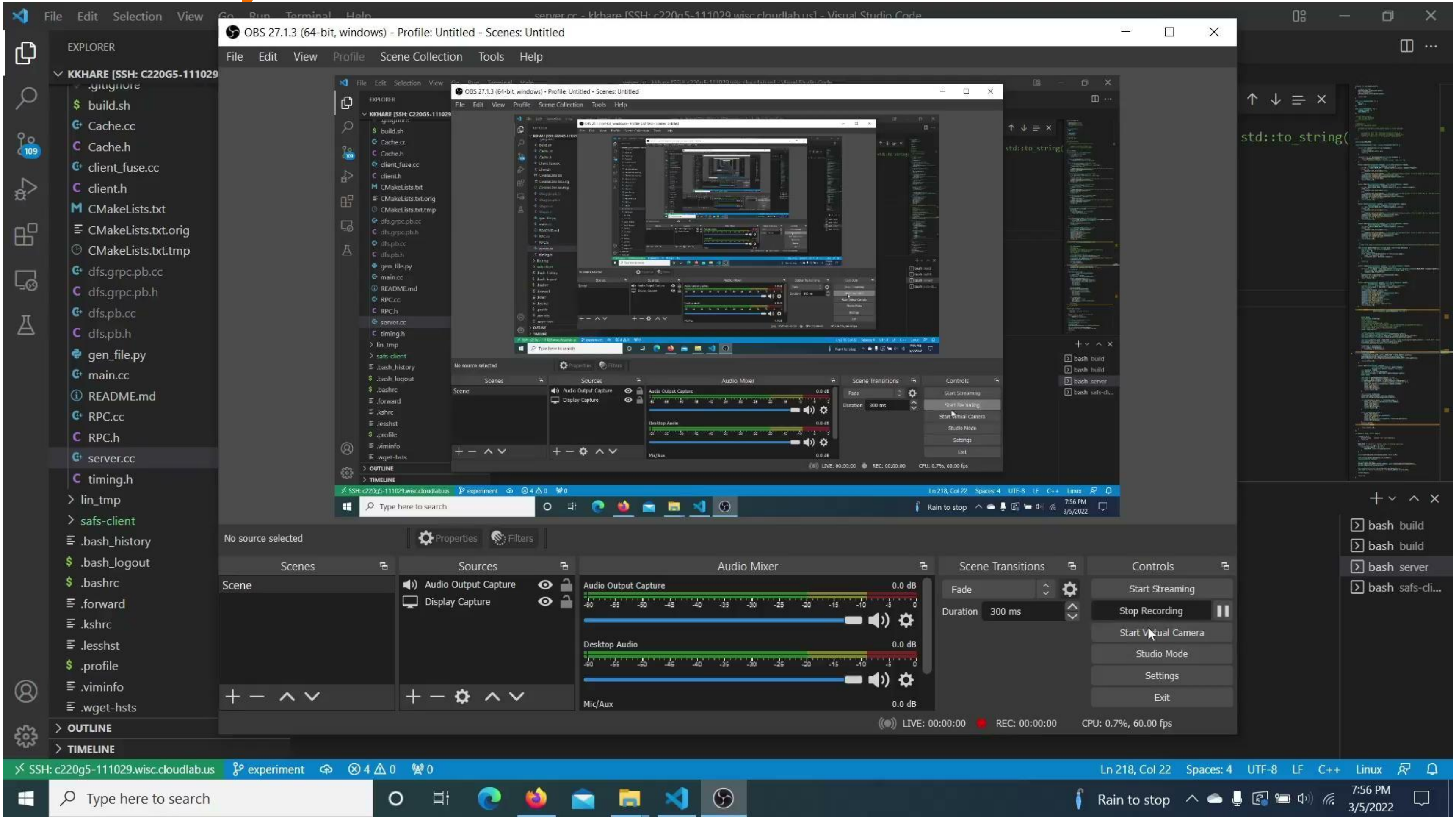
Reliability Demo - Basic AFS Protocol

Reliability Demo - Client Crash [1/2]



Reliability Demo - Client Recovery [2/2]

Reliability Demo - Server Crash



Measurements

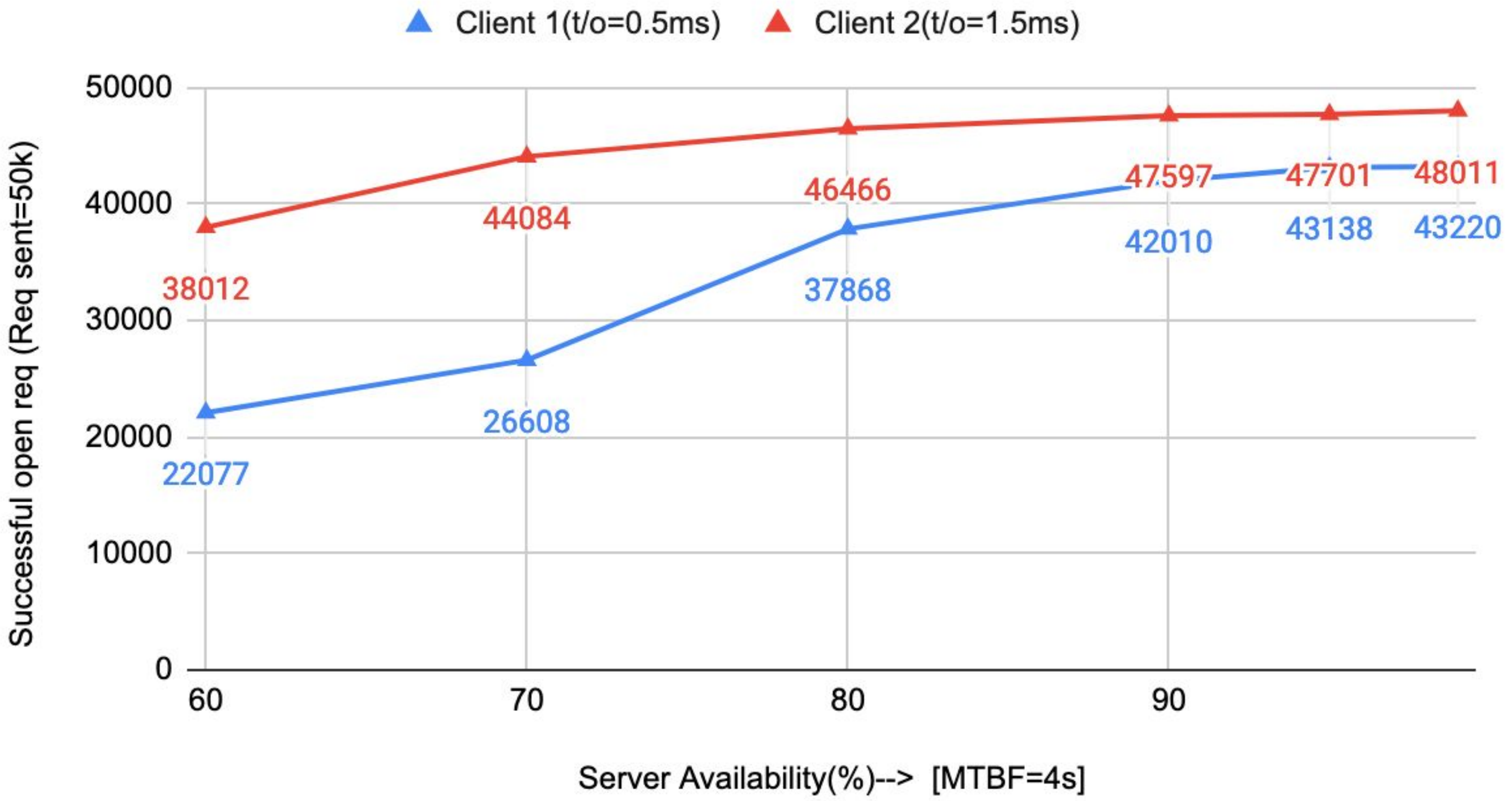
Hardware used : Cloudlab machines

Memory: 16GB

Processor: Intel Xeon Silver @2.2GHz 40 cores

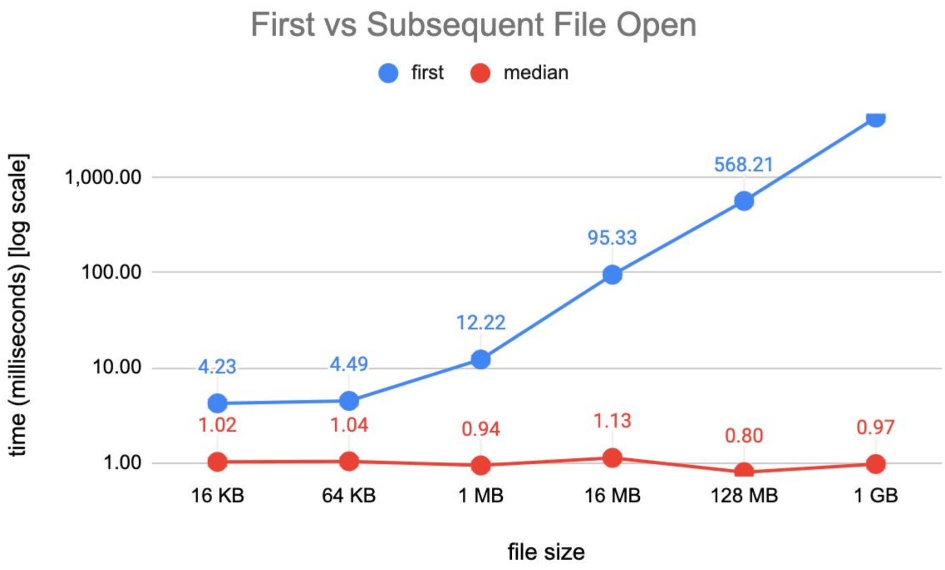
Fuse version : libfuse-dev 2.9.7

File access vs Server Availability (Different retry timeout)



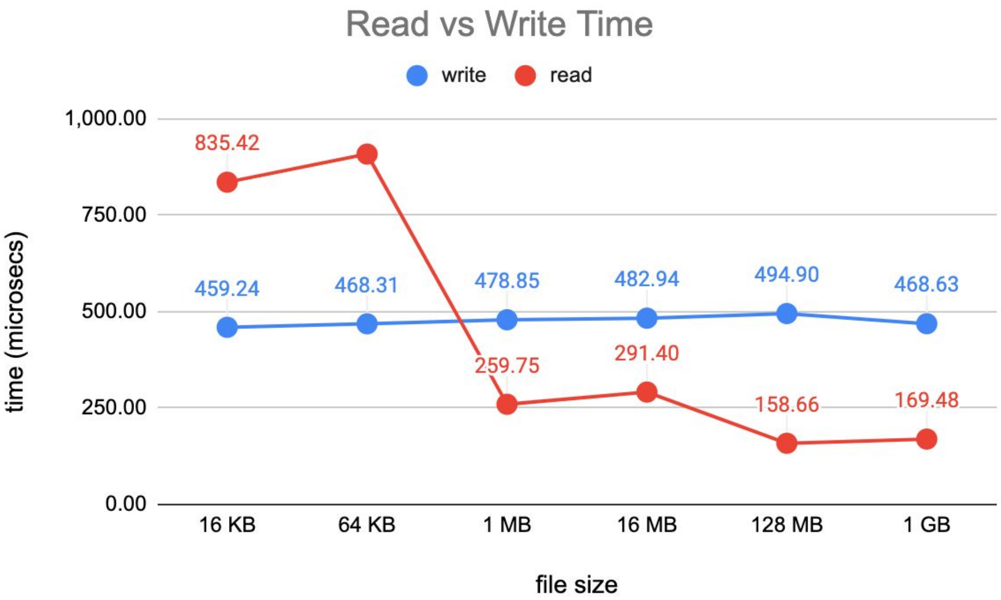
First and Subsequent File Accesses

- First accesses is slower as compared to other file accesses
- Open times don’t change for different file sizes for subsequent accesses
  - Only if the file is not fetched again from server



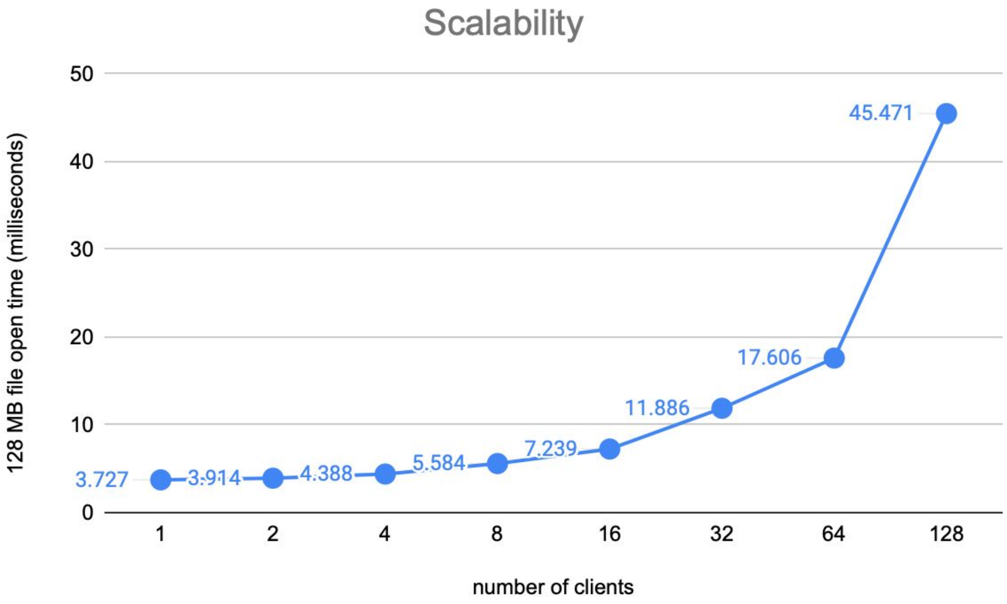
Read and Write Times

- All Reads and Writes in local
- Doesn't vary much across different file sizes
- Faster compared to Open and Close
    - No network latency in reads and write

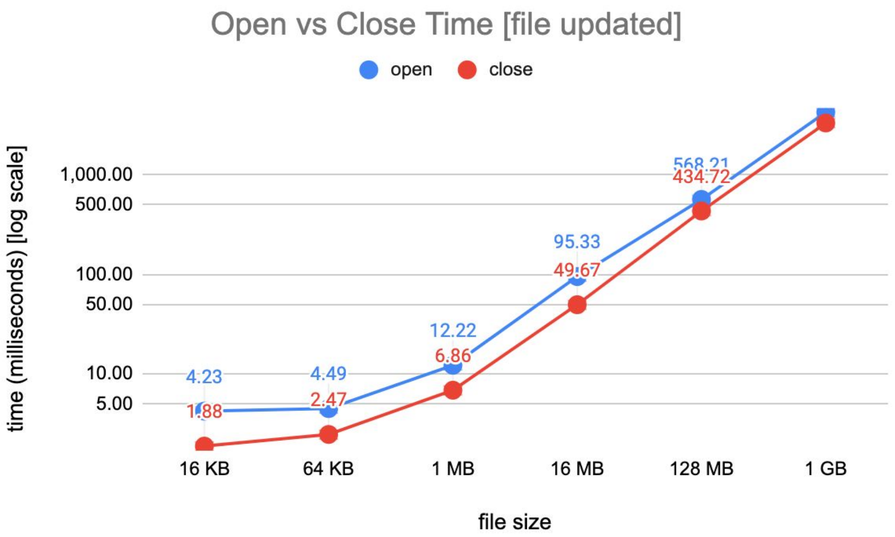


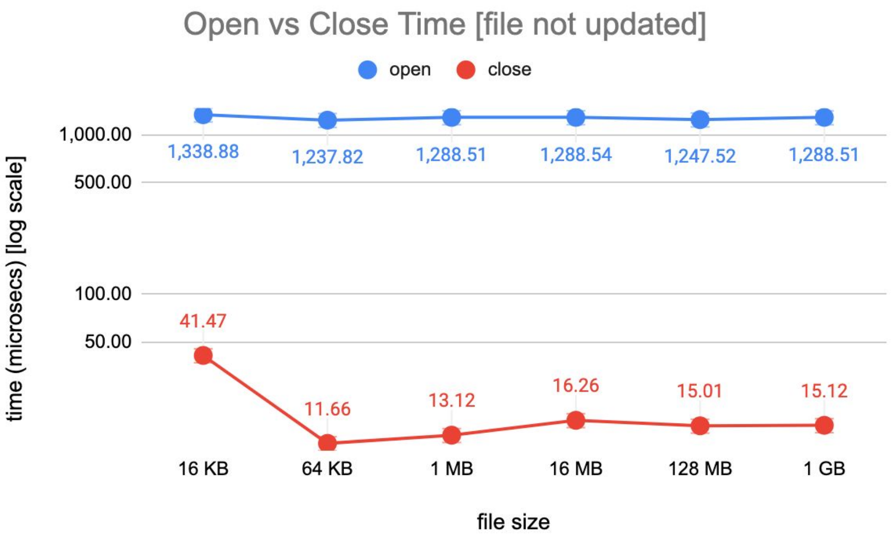
Scalability

- Multiple clients trying to open files on the server concurrently
- Takes more time as the number of clients increases



Open and Close Times





- Open and Close times increase as the file size increases only when the file is updated
  - Time high due to File Streaming

- Not much variation in open and close times across file sizes when the file is not updated
- Close faster than Open as no network overhead

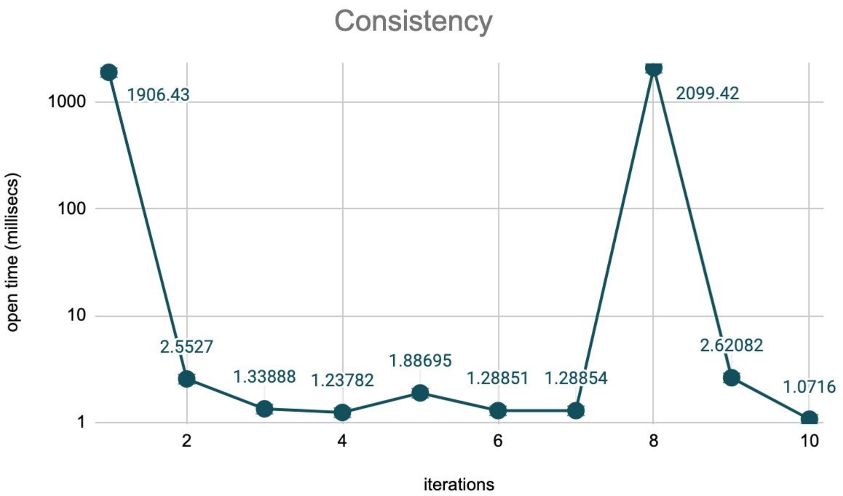
Performance Hit due to Consistency Protocol

- Time for first open is high
  - Streaming whole file

- Other Opens are faster

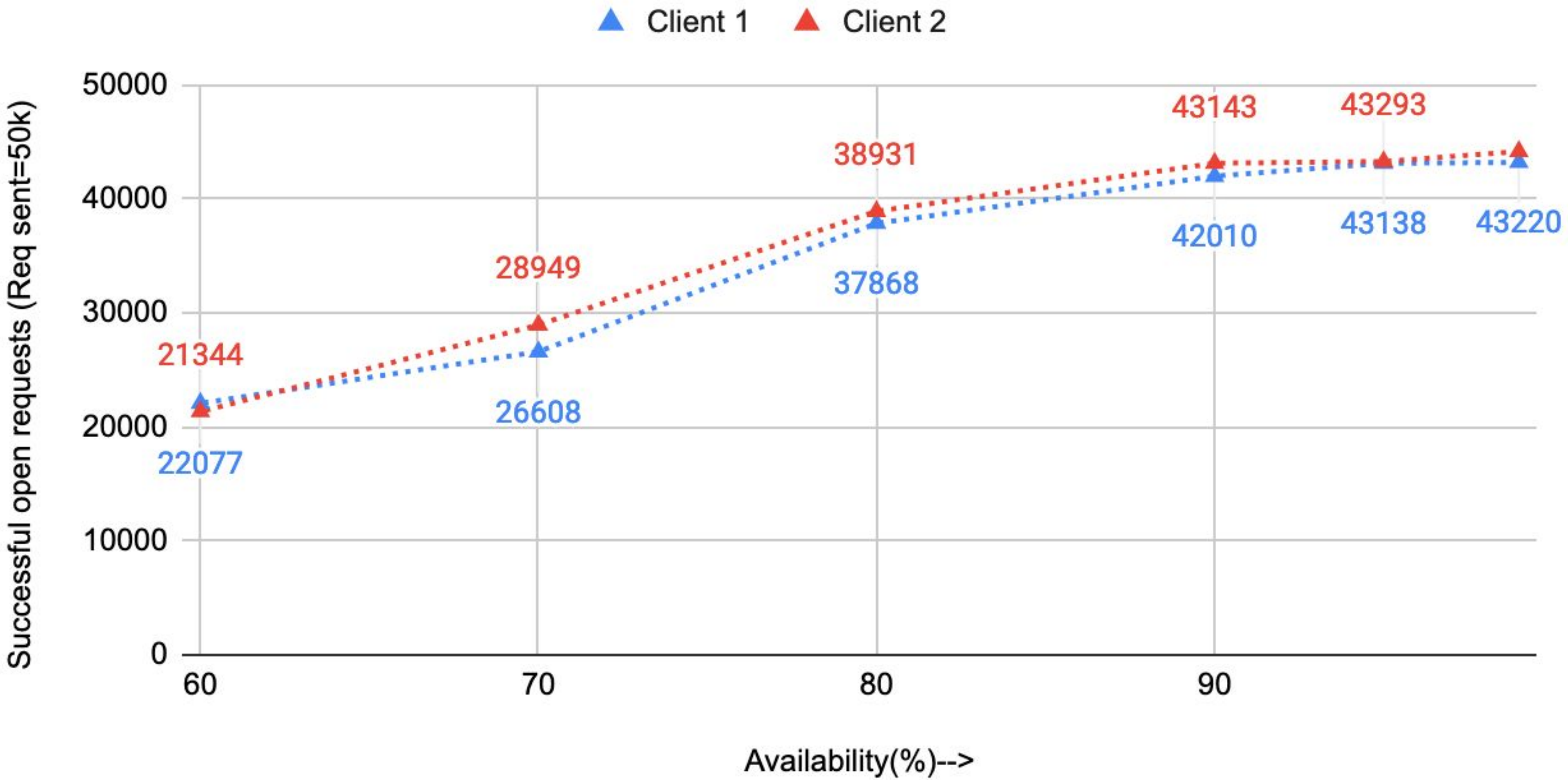
- *When file is changed in server, the client sees updated file on the next open* is one of our consistency protocols

- File changed at before 8th iteration - time is high
  - Streaming whole file



Thank you!

File access vs Availability (MTBF=4s)


Client 1
Client 2
Successful open requests (Req sent=50k)
Availability(%)-->