CS739 Project 2

Wei-Chen Chen, Lun-Cheng Chu

- AFS-like protocol
  - Implementation: grpc + FUSE
    - gcc, compile with -O3
  - Cache data types: file, attribute, cache_reply
    - File and cache_replay are stored in disk.
    - Attribute is stored in both memory and disk.
    - ex: a file in server path "/tmp/tafs/file" corresponds to client cache:
      - Client cache file: "/tmp/cache/file"
      - Client cache attribute: "/tmp/cache/file.attr"
      - Client cache cache_reply: "/tmp/cache/file.rele"
  - Support multiple clients consistency
    - Last writer wins
  - Support Crash recover
    - Once the file update is flushed, the client system will synchronize with server after reboot.
    - If crash happened before flush(), then the updates in file will disappear.

- File system functionality support:
  - getattr(), access(), readdir(), mknod(), mkdir(), unlink(), rmdir(), rename(), truncate(), open(), read(), write(), flush(), release().
  - Each function corresponds a specific funciton in FUSE.
  - Most funcitons in FUSE have corresponding grpc fucnctions.
    - write(), read() and rddir() use grpc streaming type, otherwise the data would be corrupted. The streaming size 1 mb.

- Cache Protocol:
  - A system call (read/write) usually involves serveral operations, so we properly modified FUSE functions to implement cache feature.
    - ex: a **cat** operation triggers: getattr() -> open() -> read() -> getattr() -> read() -> flush() -> release()
  - Read / Write a file
    a. getattr(): **Make sure get up-to-date file attribute**
       a. Check if there is any cache updates needed to write back to server; if so, then write back.
       b. Retrieve file attribute from server, and cache in memory.
    b. open(): **Make sure up-to-date file in cache**
       a. Retrieve file attribute from client disk, if there is any.
       b. Compare the attribute between server and client
          - If the same, then return.
          - Otherwise, retreive file from server and store both attribute and file to the client cache in disk.
    c. read(): **read from cache**
       a. read file directly from cache.
    d. flush(): **fsync**

        a. make sure the cache is saved in the disk, using fsync().
        b. apply cache reply mechanism to make sure the file updates are written back to server.
            - generate a empty file to indicate the file is not write back yet, e.g. "/tmp/cache/file.rele"
    e. release(): **write back to server**
        a. check if there is updates in file, and write back if so.
        b. update cache reply
            - e.g. remove "/tmp/cache/file.rele"
- Consistency Protocol:
    - Read / Write a file
        a. getattr():
            - Check if there is any cache updates needed to write back to server
                - e.g. check "/tmp/cache/file.rele"
        b. flush():
            - apply cache reply mechanism to make sure the file updates are written back to server.
                - generate a empty file to indicate the file is not write back yet, e.g. "/tmp/cache/file.rele"
        c. release():
            - write back to server
            - update cache reply
                - e.g. remove "/tmp/cache/file.rele"
- Crash recover
    - Before the operation reaches flush(), i.e. save to disk, the updates to the file is in the memroy. If crash happens before flush(), we will loss the updates. Once the updates is flushed to cache in disk, and the client crashs before write back to server, we can recover the updates to server when client is back.

- Improve performance
    - attribute: put attribute in memory cache for fast access
    - Pass file descriptor instead of open a file to reduce the overhead. Will close it in flush().

TODO:
Use cache data for 2nd write

Read

1st **read**	2nd **read**
getattr() 1. fetch attr from server	**getattr()** 1. fetch attr from server
open() 1. no cached attribute 2. fetch from server (file and attr) 3. store in cache (disk)	**open()** 1. compare attribute between server and cached attr 2. same attr, return
read() 1. read file from cache (size, offset)	**read()** 1. read file from cache (size, offset)
getattr() 1. fetch attr from server	**getattr()** 1. fetch attr from server
flush() 1. fsync cache to disk	**flush()** 1. fsync cache to disk
release()	**release()**

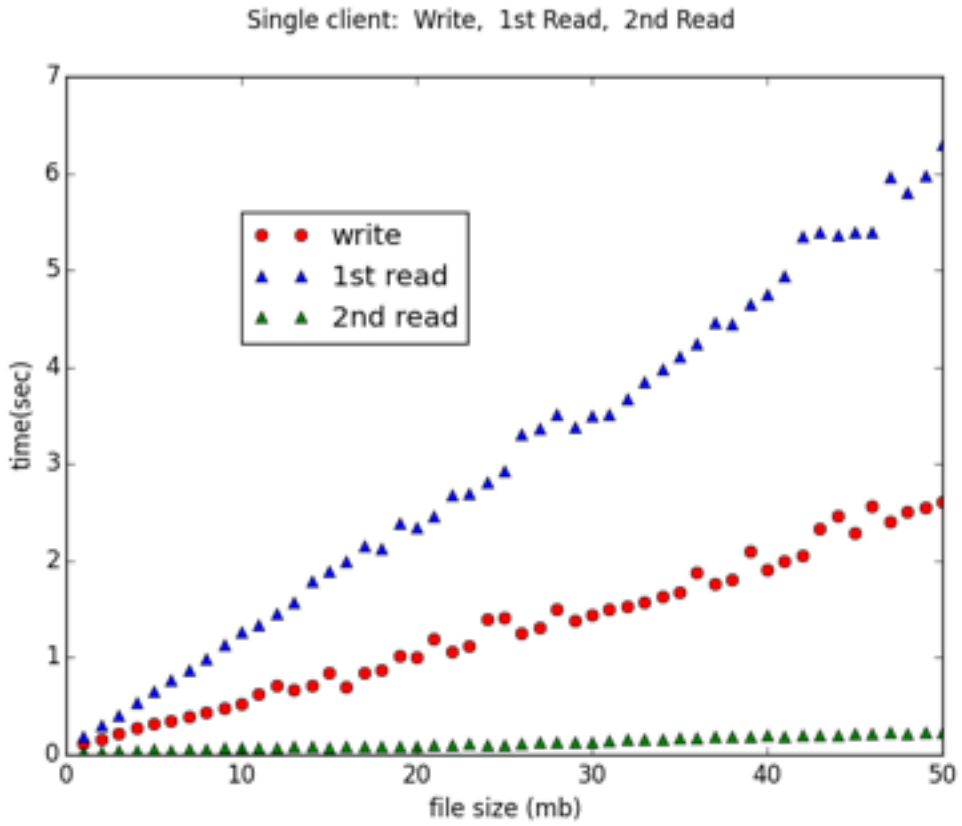
Write

1st **write**	2nd **write**
getattr() 1. fetch attr from server	same as 1st write (pure write)
mknode() 1. make node in server	
getattr()	
open()	
flush()	
write() 1. update file in cache	
flush() 1. fsync cache to disk 2. add flag in disk to notify write back to serve (cache reply)	
release() 1. write back to server 2. remove flag	

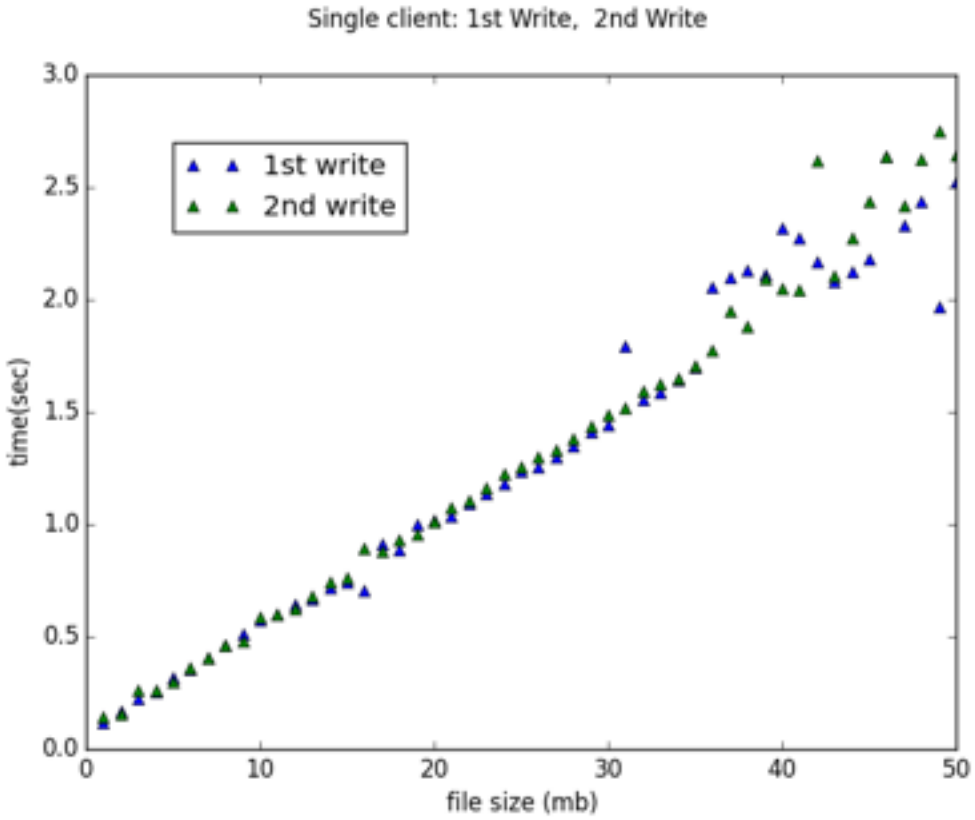
1st Read then **Write**

1st **read**	**write** (create a new file)
getattr() 1. fetch attr from server	**getattr()** 1. fetch attr from server
open() 1. no cached attribute 2. fetch from server (file and attr) 3. update file in cache (disk)	**mknode()** 1. make node in server
flush()	**getattr()**
read() 1. read file from cache (size, offset)	**open()**
flush() 1. fsync cache to disk	**flush()**
release()	**write()** 1. update file in cache (disk)
	flush() 1. fsync cache to disk 2. add flag in disk to notify write back to serve (cache reply)
	release() 1. write back to server 2. remove flag

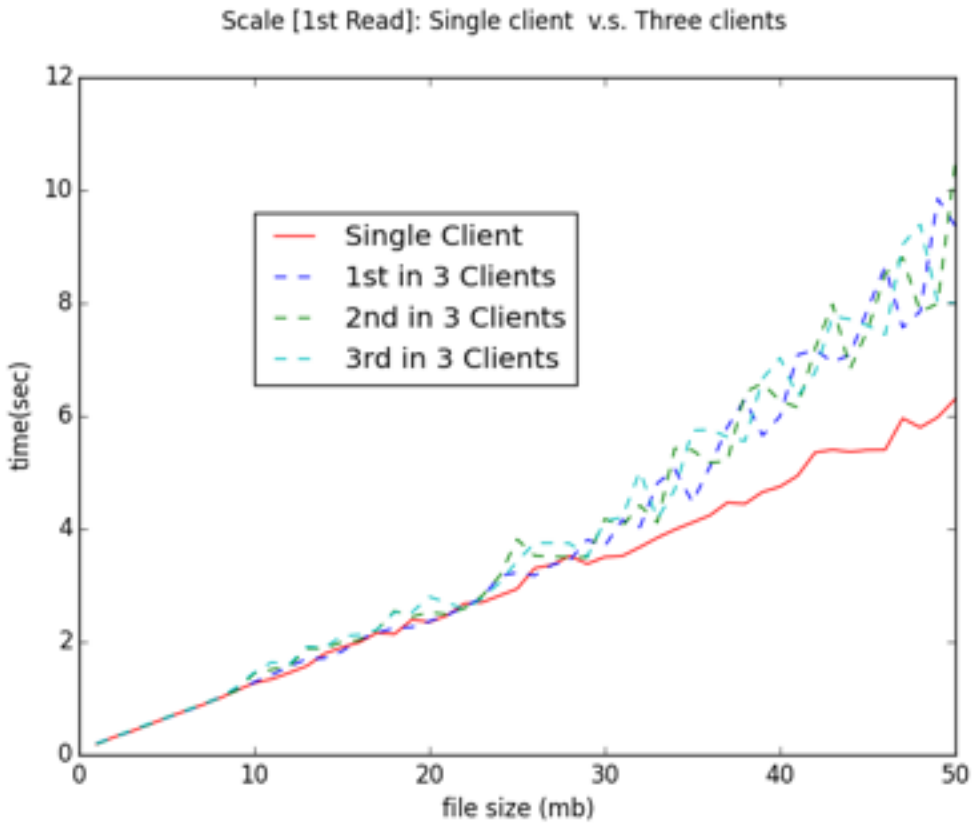
1. Single Client: 1st **write**, 1st **read**, 2nd **read**



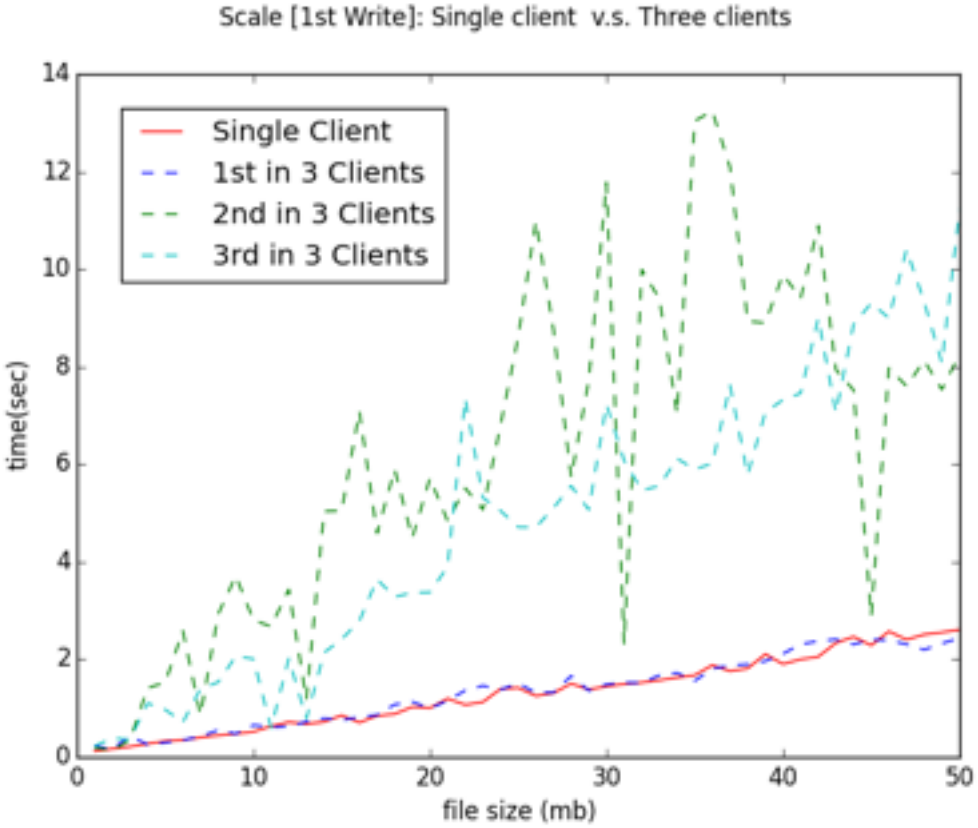
2. Single Client: 1st **write**, and 2nd **write**



3. Three clients: 1st Read



4. Three clients: 1st Write



5. Consistency protocol