(function code)

Upd_server.py

- **Leader election**: lowest ID leader default
  leader sends periodic heartbeats to followers `send_heartbeats`
  followers monitor heartbeats `process_heartbeat`
  if none received within election timeout tries to become leader, but concedes if it receives heartbeat from lower id server

- **Fault tolerance**: persistent state saved to local json file
  ∴ recover state upon restart
  followers sync loc states and file updates from leader `sync_file` `sync_lock_state`
  `notify_followers_lock_state` `_file_update`

- **Consistency and Safety**:
  - Locking → grants lock to one client at a time
    → enforces time based lock expiration
    → queued clients
  - Duplicate requests: tracking client modifications to avoid processing duplicate commands
    `modif_requests`

- **Message handling**
  → Port 8000: Client-server communication `handle_requests`
  - acquire / release lock
  - append file
  - identify leader

• client only communicates with leader
• client request go to queue (client may check request queue status)

  → Port 8001: server-replica communication `handle_manager`
  - heartbeat
  - lock state: only granted by leader
  - file update: only granted by leader

- **Crash mitigation**
  → Network: • heartbeat timeouts trigger leader re-election
  • retry mechanisms for key actions
  → Client errors: • clients receive explicit responses for invalid requests "You do not hold lock"
  • queue request avoids race conditions
  → Server crash: • followers rely on leader for consistency
  • leader election ensures there is always a leader

RPC_connection.py

timeouts / retries / exponential backoffs

Distributed_Client.py

1) Finds leader - only communicates with leader
find_leader — other servers get sent replica commands from leader

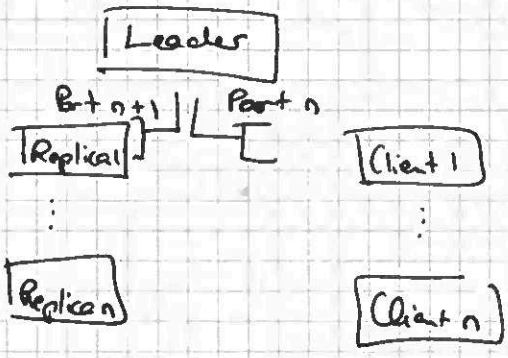
2) Send task

2.1) Acquire lock : sends request message, receives queue position
(timeout and "wrong" leader responses also possible)
acquire_lock → lock
↓
start lock expiration
lock start_lease_timer

2.2) Sends message : "append_file"
append_file

2.3) Waits for reply

Heartbeats between client and server track lease.



Client(s) —Who is leader?→ Server(s)

X leader

requests to X ⟷ Response

X sends message to replicas

- queue
- lock acquired
- lock released

Limitations : 1) I was not able to implement strong consistency for leader election, as such I cannot run the advanced test cases.

2) My queue and lock timeout "breaks often"* For this reason fails:
- client_failure_could_after_edit
- single_server_failure_lock_free
- single_server_failure_lock_held

Fail.

* the server fails to properly transfer lock to the next client in queue after the lock expires, causing client timeout. In the 3 failing test operations are not necessarily completed at/close of lock lease duration which also requiring queue handling, though the this shortcoming of the servers doesn't appear.