Skinny

a simplified implementation of Chubby

Cody Tseng

Ping-Han Chuang

Wuh-Chwen Hwang

Content

- Design
  - Features
  - RPC call
  - System structure
- Experiment Setup
- Performance Measurement
- Test and Demo

Design – Features

- Aim to be as feature complete as possible
- Supported Features
  - Client session
  - File open/close/delete
  - File read/write
  - Reader/writer lock
  - Single kind of event
  - Directory
  - Client-side Cache
  - Ephemeral files
- Unsupported Features
  - ACL, various kinds of event, snapshot, proxies, lock sequencer and lock upgrading

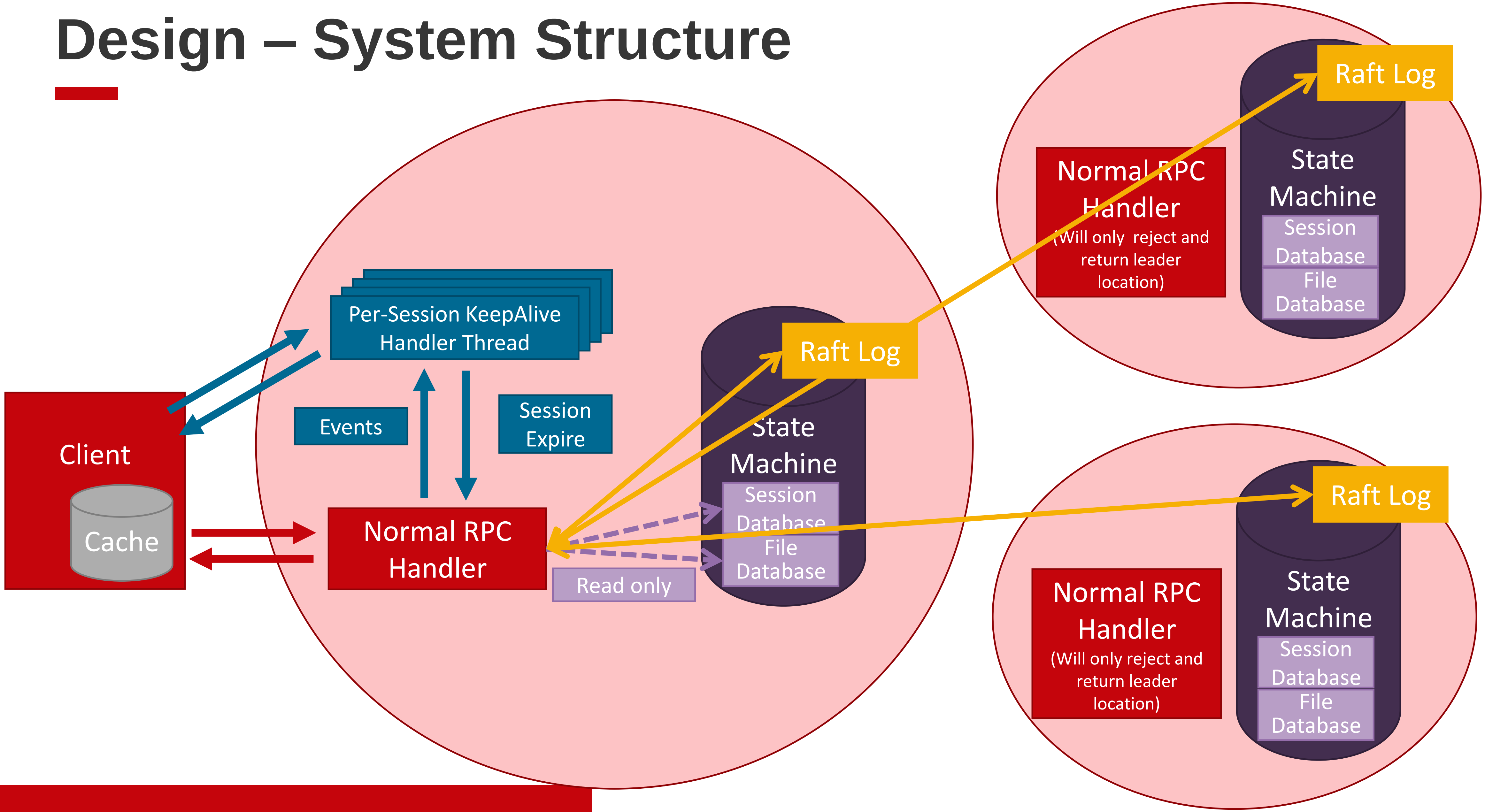
Design – System Structure

Server

- 3 or 5 servers in a Skinny cell
  - ebay/NuRaft library
  - In-memory states

- Client Library
  - Send keep alive
    - Occasionally receive event and cache invalidation request

Design – System Structure



Design – Files, directories, and handles

- Identify via path name. E.g. `/hello/world`
- Per file metadata
  - **`int instance_num`**
  - `bool file_exists`
  - `bool is_directory`
  - **`bool is_ephemeral`**
  - `bool is_locked_ex`
  - `lock_owners;`
  - `opened_Session;`
- File handle
  - Unlike Chubby, client sees file handle as simply an integer
  - Session DB saves instance number

Design - Locks

- Reader-writer lock
- Blocking & non-blocking semantics
- Lock sequencer and lock delay not implemented

- `(bool) TryAcquire(fh, ex)`
- `() Acquire(fh, ex)`
- `() Release (fh)`

Design - Events

- User can specify event callback when Open()
- Piggyback in keepalive response
- Client lib invokes callback when:
  - File: modified / deleted
  - Directory: contained file(s) added / removed
- `(fh) Open (path, callback, is_ephemeral)`
- `(fh) OpenDir (path, callback, is_ephemeral)`

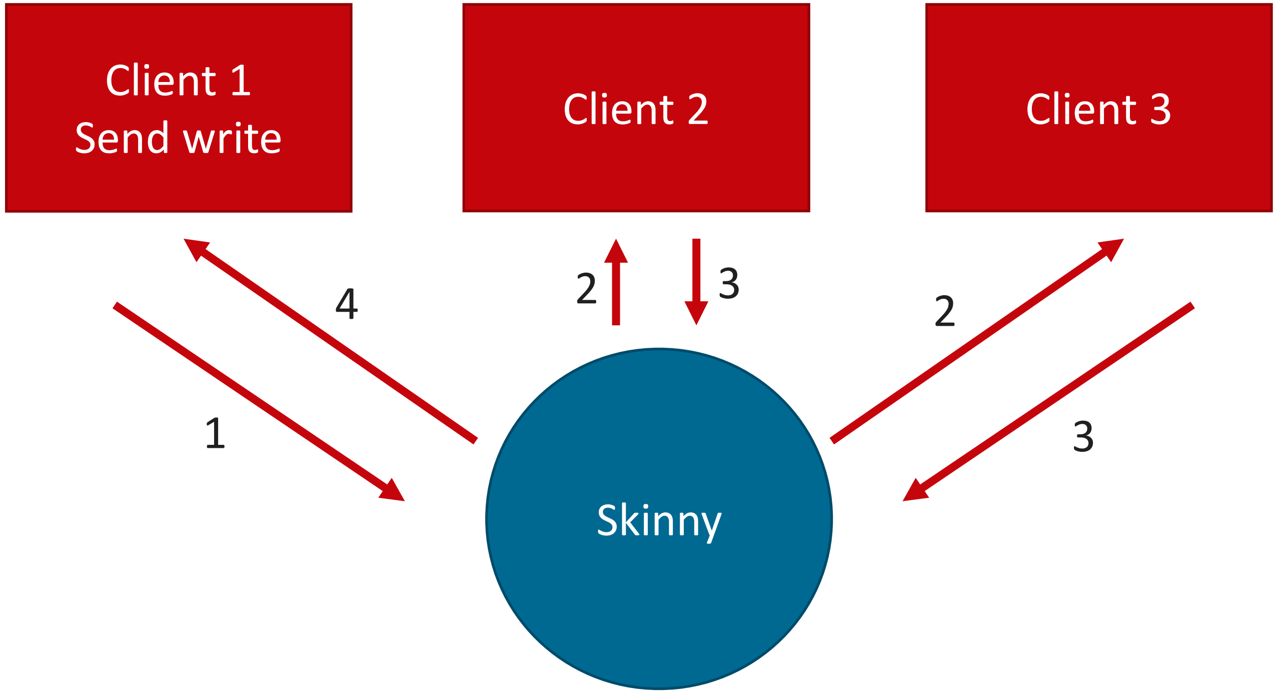
Design – Client API

Comply with Chubby: every API except Open(Dir) uses fh

- (constructor)[implicitly called StartSession()]
- (destructor)[implicitly called EndSession()]
- (fh) Open (path, callback, is_ephemeral)
- (fh) OpenDir (path, callback, is_ephemeral)
- () Close(fh)
- (content) GetContent(fh)
- () SetContent(fh, content)
- () Delete(fh)
- (bool) TryAcquire(fh, ex)
- () Acquire(fh, ex)
- () Release (fh)

Design - Caching

- Client will cache data content
- Server invalidate cache when handling write
- Writes will not return until all session acked or expired



Design - Session + KeepAlive

- Client:
  - Must send KeepAlive periodically to maintain an active session.
  - If server does not response in 10s, mark the session as expired
    - Subsequent calls throw error
  - Invalidate cache if KeepAlive returned with a file handle
  - Event: call callback if the user previously register an event associate with that file handle

Design - Sessions + KeepAlive

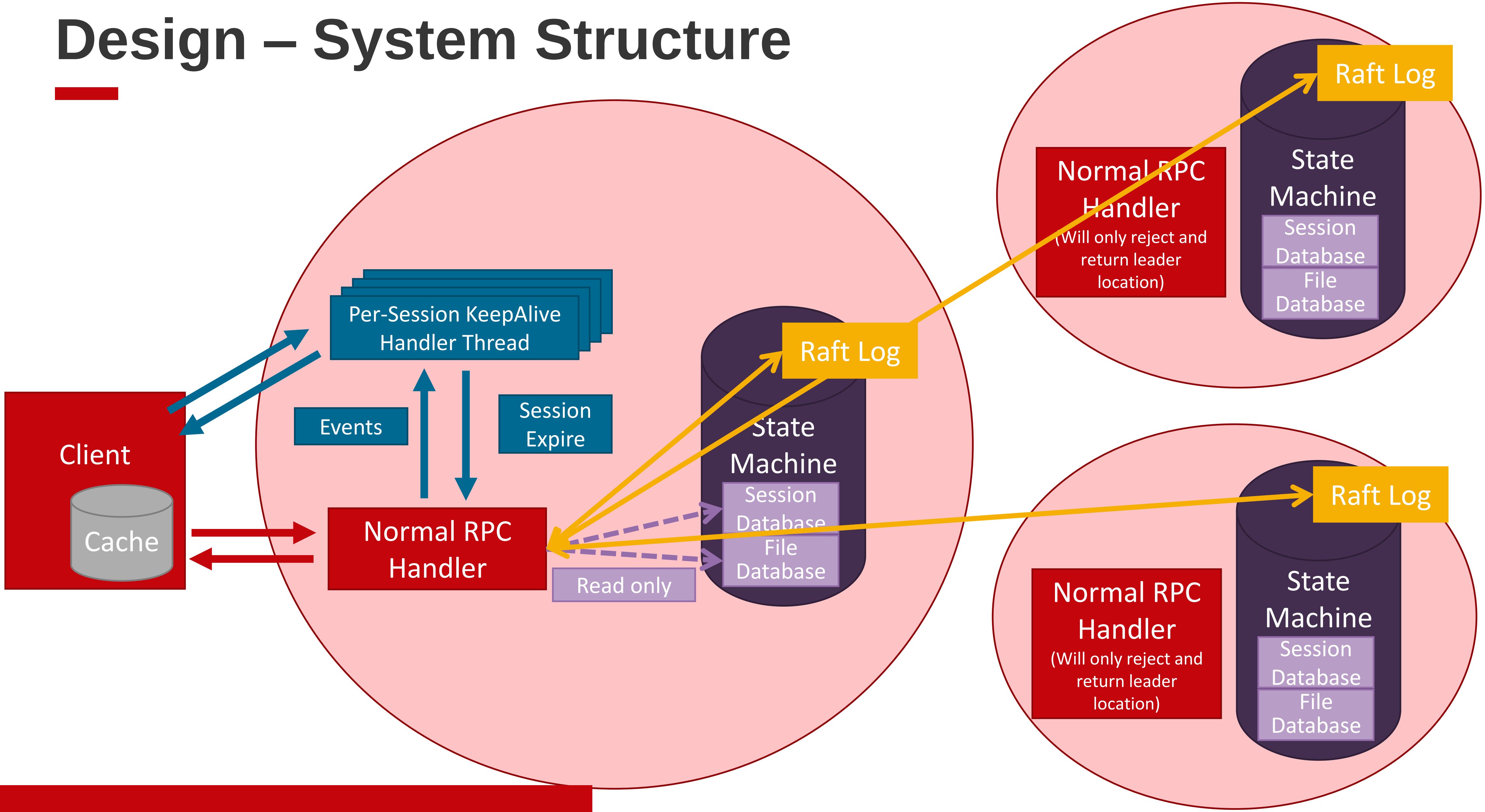
Server: per-session KeepAlive handler thread

- Thread wakes up due to:
    - Receive a new KeepAlive request within timeout => replace current request
        - If there is an event in queue, return immediately w/ message
        - Otherwise, wait 5 seconds, then reset timeout, return call, sleep
    - Receive end_session: cancel and clean up
    - Receive a new event: put in queue
    - Timeout: Kill session
- Timeout: 5s

Leader Failure

- Client – Clear cache, block all requests
- Server – Elect new leader, start sessions’ KeepAlive threads (imply timer reset)

Design – System Structure



Correctness Testing

- Bind C++ client library to Python code
- Write test suites in Python (easier to write)
- Able to launch/kill servers in Python
- Test locks/reads/writes with 1000s of clients, events, leader failure, etc.
- Found multiple bugs, mostly race condition

Correctness Testing

- Read/write
  - Single writer, many writes
  - Single writer, one write, many readers
- Lock
  - Single user Acquire, TryAcquire, Release; Acquire after Release
  - Multiple users contending (non-blocking) the same lock
  - Multiple users wait **(blocked)**, acquire, and release the same lock
  - **Single writer Acquire** after multiple readers Release
  - **Multiple readers Acquire** after a single writer Release

Correctness Testing

- Event
  - **Event callback** invocation
  - Distributed **barrier** (used in performance measurement)
- Cache
  - Cache invalidation under client failures
- Leader failure

Performance Measurement

Hardware

- Cloudlab c8220 - 40 logical cores, 256 GB RAM

Server: 3/5 nodes cluster

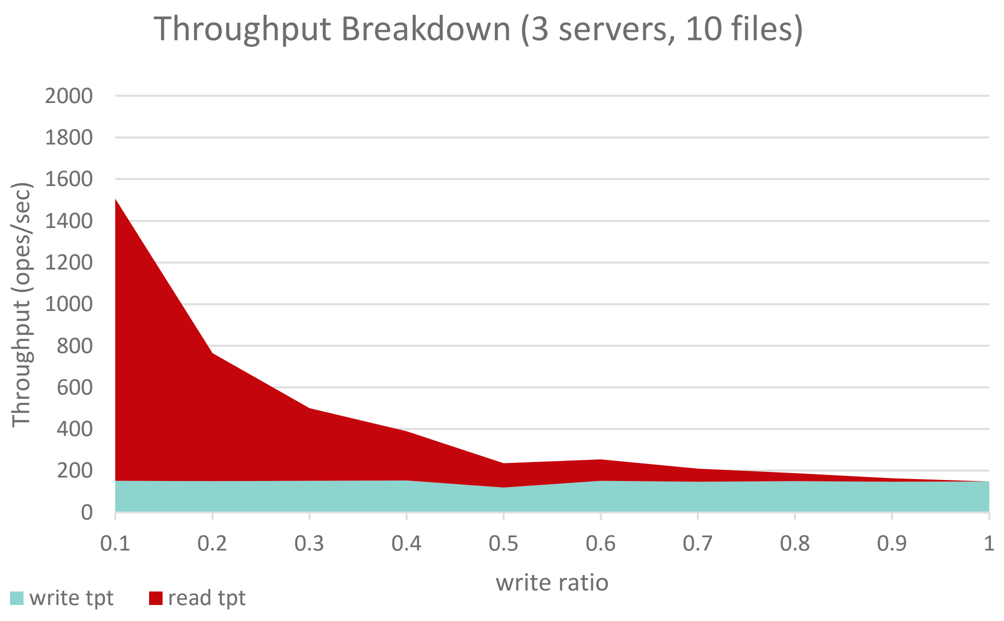
Client (C++)

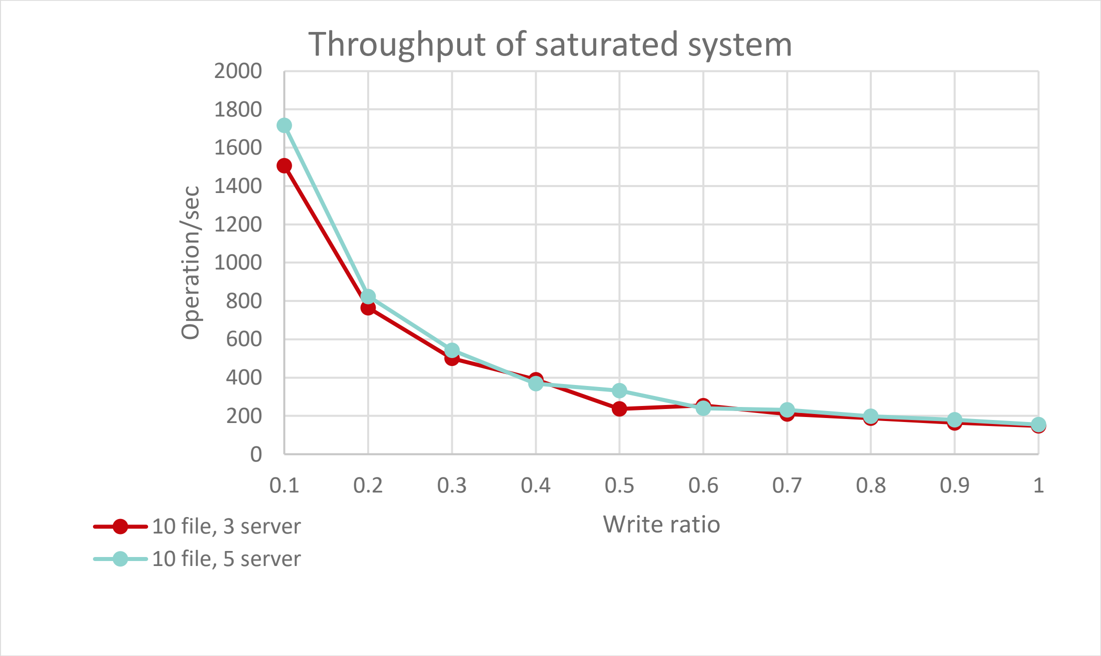
- 5 nodes, each runs 30 clients
- Open certain amount of files
- Start simultaneously (barrier by Skinny)
- Keep issuing requests to those files for 30 secs
- Each request: randomly [pick a file] & [read or write]
- Parameter:
  - # files
  - Request read/write ratio
- Measurement: throughput (ops/sec), latency

Performance Measurement - Throughput

- Throughput decreases as write ratio increases
  - # reads drop, # writes keep the same
  - Write is the bottleneck

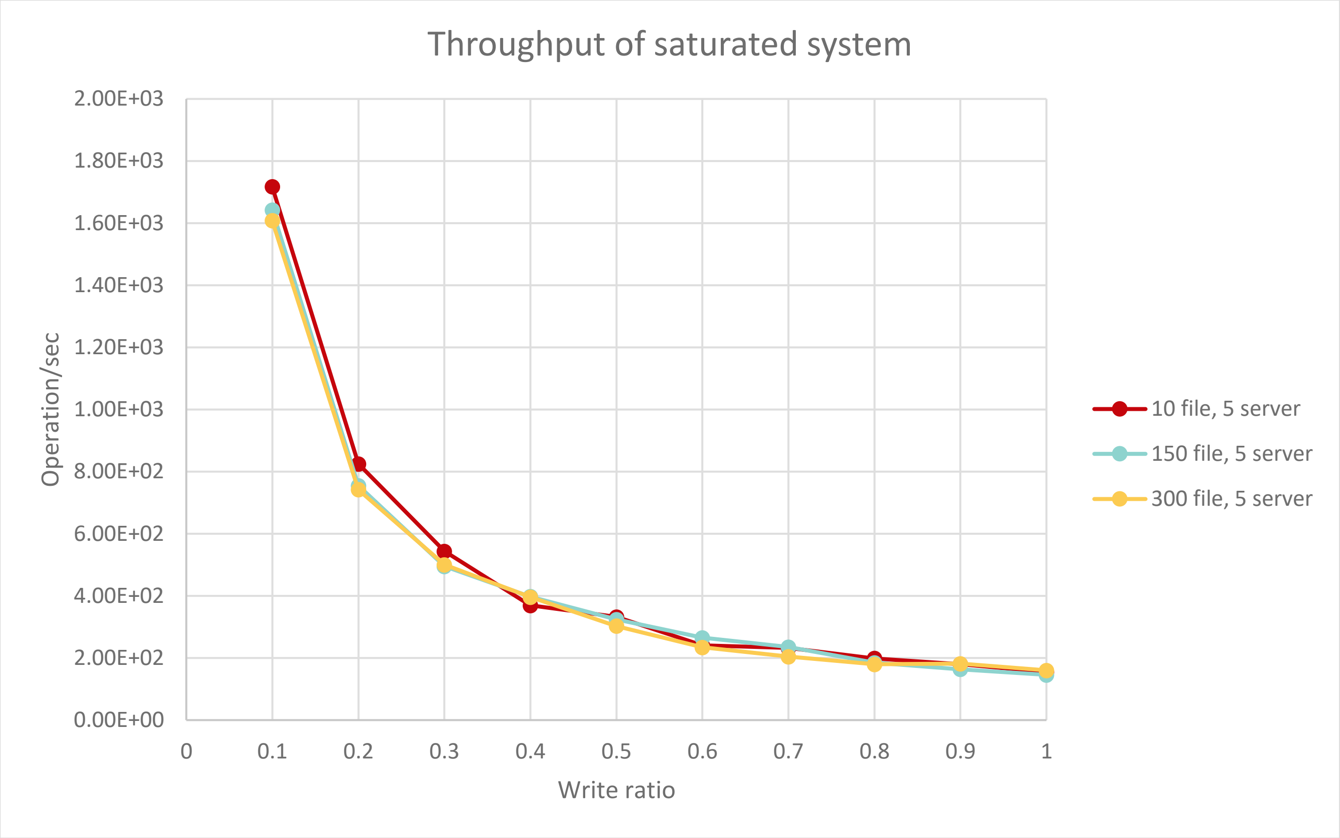
- Number of servers does not affect the result much
  - Servers sit in the same data center





Performance Measurement - Throughput

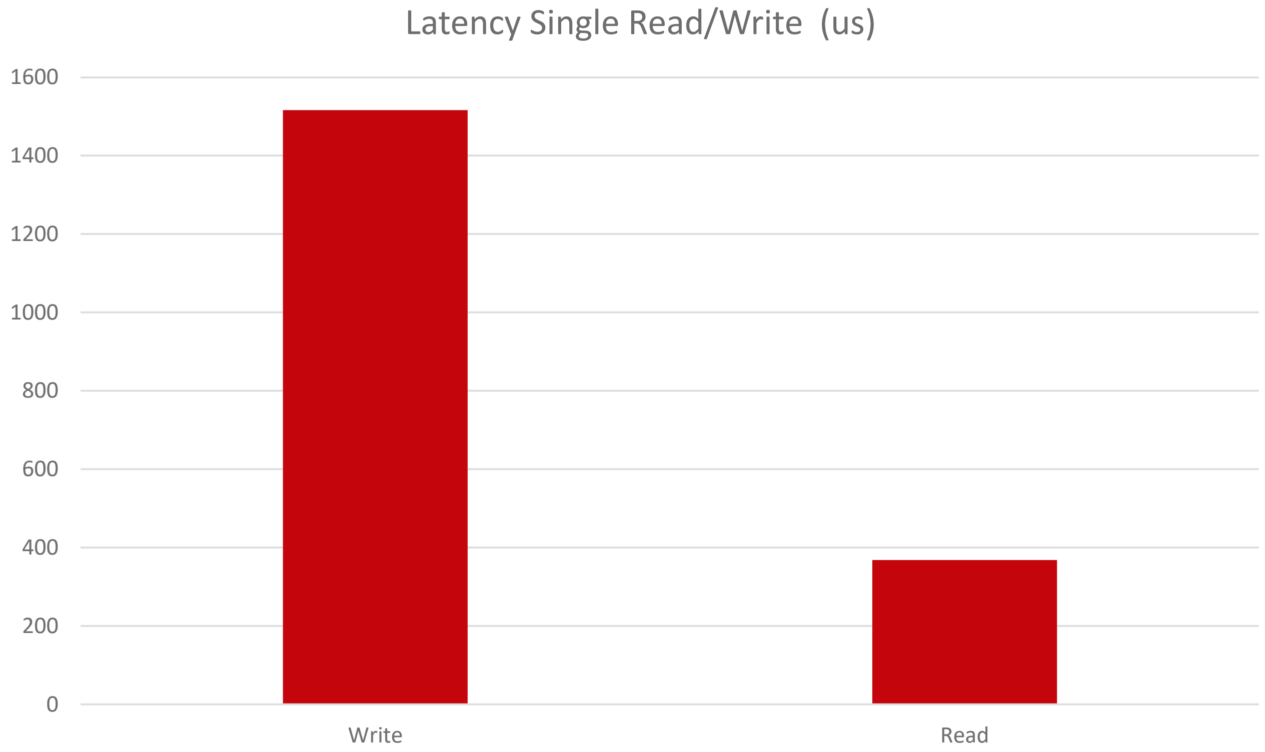
- Changing # opened files
  - Hypothesis: lower # files => higher chance of cache invalidation on the file to be read
  - Result: does not affect much
  - Probably need much more files
  - Opening files takes too much time in Skinny



Performance Measurement – Average Latency

- Single Client
- Single Request

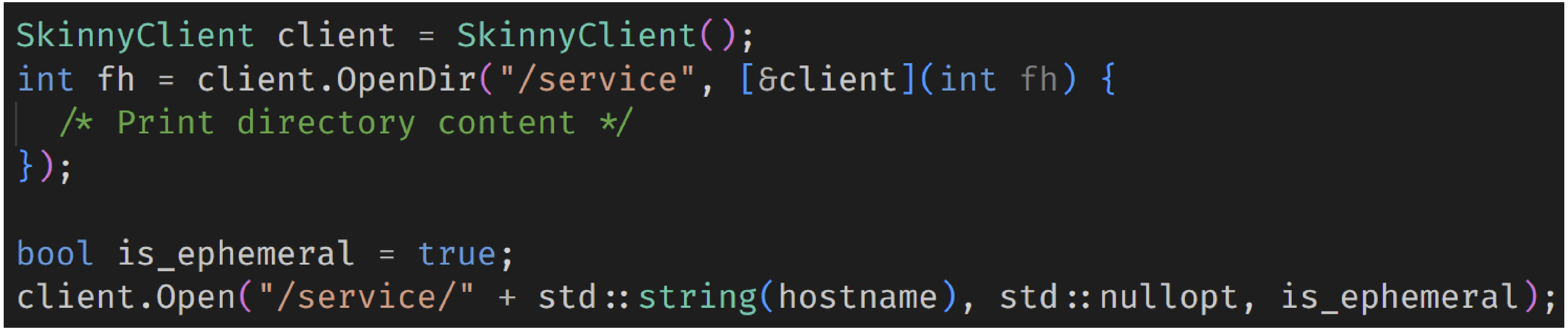
Write	1516.57 us
Read	368.163 us



Demo 1 – Basic functionalities & Leader crash

- Basic functionalities – Open / Read / Write / Lock
- Show that previous content can still be read even if the leader crash
- Show that lock status is still maintained after a leader crash

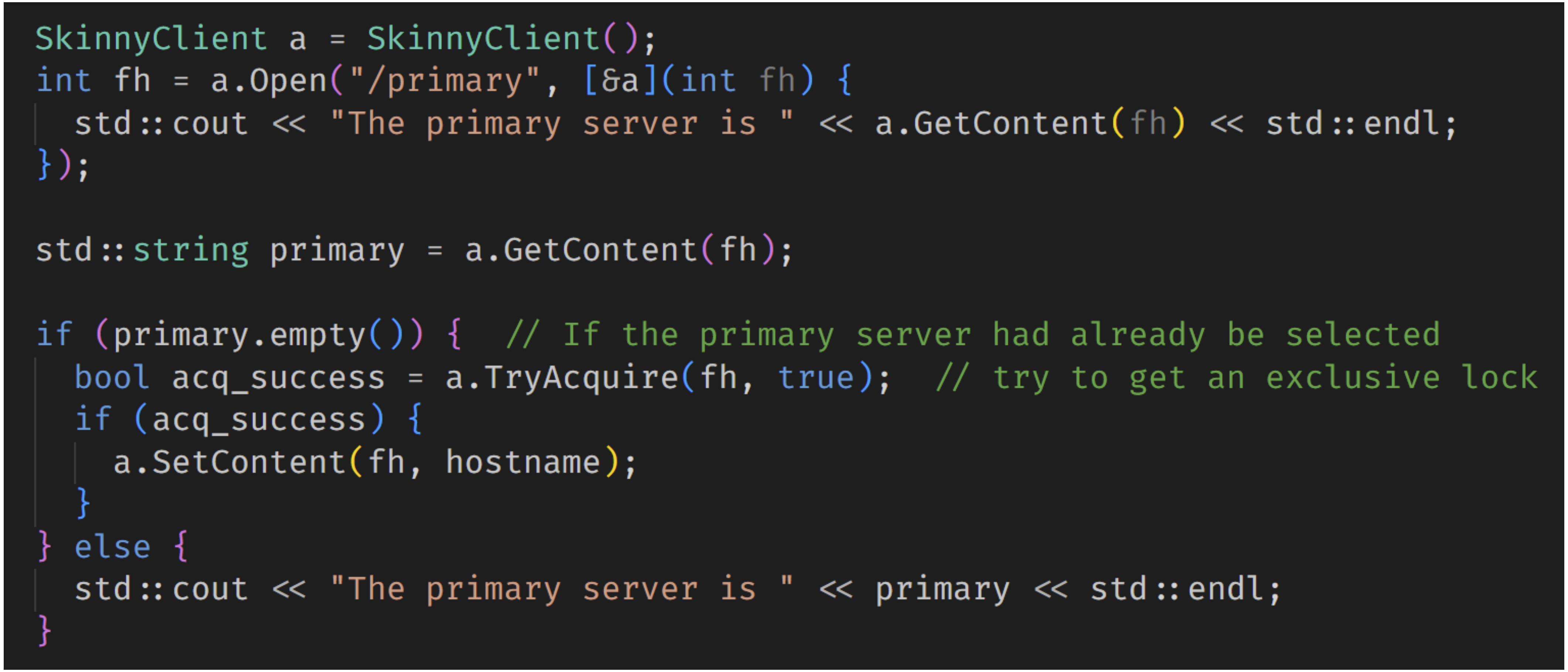
Demo 2 – Membership changes



```
SkinnyClient client = SkinnyClient();
int fh = client.OpenDir("/service", [&client](int fh) {
  /* Print directory content */
});

bool is_ephemeral = true;
client.Open("/service/" + std::string(hostname), std::nullopt, is_ephemeral);
```

Demo 3 – Primary Election



```
SkinnyClient a = SkinnyClient();
int fh = a.Open("/primary", [&a](int fh) {
  std::cout << "The primary server is " << a.GetContent(fh) << std::endl;
});

std::string primary = a.GetContent(fh);

if (primary.empty()) {  // If the primary server had already be selected
  bool acq_success = a.TryAcquire(fh, true);  // try to get an exclusive lock
  if (acq_success) {
    a.SetContent(fh, hostname);
  }
} else {
  std::cout << "The primary server is " << primary << std::endl;
}
```

Lessons Learned

- Should’ve clearly defined where to put logic
  - State Machine vs GRPC Service
- State Machine operations should never block
- Should’ve used some library that provide synchronized container
  - `Synchronized<vector<int>> vec;`
  - `vec.wlock()->push_back(3)`