UNIVERSITY OF SCIENCE, VNU-HCMC

FACULTY INFORMATION TECHNOLOGY

Blockchain và ứng dụng - 22MMT

BÁO CÁO LAB 1

Instructors: Nguyễn Đình Thúc, Ngô Đình Hy, Nguyễn Ngọc Toàn, Nguyễn Văn Quang Huy

Class: 22MMT

Members:

22127210 – Phạm Anh Khôi

22127092 – Lê Bảo Giang

22127133 – Đinh Vũ Huân

20127656 – Trần Minh Trường

21127594 – Phạm Huỳnh Tấn Đạt

Ho Chi Minh City – 2025

Mục lục

RAFT

1 Tổng Quan

RAFT được thiết kế để quản lý nhật ký sao chép (**replicated log**) trong hệ thống phân tán với mục tiêu tối thượng là **tính dễ hiểu**. Thuật toán tập trung vào khả năng chịu lỗi dừng (**crash failures**) để đảm bảo một cụm máy tính hoạt động thống nhất.

2 Cách RAFT Đảm Bảo Tính Nhất Quán (Consistency)

RAFT phân rã bài toán đồng thuận thành các thành phần chính:

- **Bầu chọn Leader:** Sử dụng **randomized election timeouts** (150-300ms) để ngăn chặn và giải quyết nhanh tình trạng chia phiếu (**split votes**). Ứng viên nhận được đa số phiếu (n/2+1) sẽ trở thành Leader.
- **Sao chép nhật ký:** Leader nhận yêu cầu từ Client, ghi vào nhật ký cục bộ và gửi **AppendEntries RPC** đến các Follower . Khi đa số các nút đã sao chép thành công, mục dữ liệu được coi là **Committed**.
- **Khớp nhật ký (Log Matching):** Leader sử dụng prevLogIndex và prevLogTerm để kiểm tra tính nhất quán trước khi chèn mục mới. Nếu có sai lệch, Leader giảm nextIndex và gửi lại dữ liệu để đồng bộ hóa

Bảng: 5 đặc tính an toàn của RAFT

Đặc tính	**Mô tả chi tiết**
Election Safety	Đảm bảo trong mỗi nhiệm kỳ (Term), tối đa chỉ có duy nhất một Leader được bầu chọn.
Leader Append-Only	Leader không bao giờ thực hiện việc ghi đè hoặc xóa các mục nhật ký, chỉ cho phép thêm các mục mới vào cuối nhật ký.
Log Matching	Nếu hai bản nhật ký có mục dữ liệu trùng khớp về chỉ số (Index) và nhiệm kỳ (Term), thì toàn bộ các mục trước đó trong hai nhật ký này là hoàn toàn giống nhau.
Leader Completeness	Bất kỳ mục nhật ký nào đã được xác nhận cam kết (Committed) trong một nhiệm kỳ thì phải hiện diện trong nhật ký của các Leader ở mọi nhiệm kỳ kế tiếp.

State Machine Safety	Nếu một nút đã áp dụng một lệnh tại một chỉ số vào máy trạng thái của nó, thì không có nút nào khác trong cụm được áp dụng một lệnh khác cho cùng chỉ số đó.

- **Ràng buộc bầu cử:** Một nút chỉ bầu cho ứng viên có nhật ký cập nhật hơn (**Up-to-date**): có nhiệm kỳ cuối cao hơn hoặc có cùng nhiệm kỳ nhưng nhật ký dài hơn.

3 Hạn Chế của RAFT với Hành Vi Độc Hại

RAFT không thể chống lại các hành vi độc hại (**malicious behaviors**). Các lỗ hổng chính bao gồm:

- **Dữ liệu không nhất quán:** Leader Byzantine có thể gửi các mục nhật ký khác nhau cho các Follower khác nhau.
- **Gian lận bầu cử:** Ứng viên có thể bỏ phiếu nhiều lần (**double voting**).
- **Tấn công thời gian:** Nút độc hại lạm dụng cơ chế timeout để liên tục kích hoạt bầu cử giả mạo (DoS)

Kết luận và Hướng giải quyết

Vì RAFT chỉ đảm bảo an toàn trong mô hình lỗi phi Byzantine (non-Byzantine environment), đối với các hệ thống yêu cầu bảo mật cao hoặc hoạt động trong môi trường không tin cậy (như mạng ngang hàng hoặc Blockchain), các giao thức chịu lỗi Byzantine như pBFT (Practical Byzantine Fault Tolerance) hoặc các biến thể của nó sẽ được ưu tiên sử dụng.

RAFT và pBFT

1 Sơ Lược Về pBFT

Giao thức Practical Byzantine Fault Tolerance (pBFT) được giới thiệu bởi Miguel Castro và Barbara Liskov vào năm 1999. Đây là một bước đột phá trong hệ thống phân tán, cung cấp giải pháp thực tiễn để đạt được sự đồng thuận trong môi trường có các nút lỗi Byzantine.

Các mục tiêu thiết kế cốt lõi:

- Xử lý Byzantine failures (malicious nodes)

- Khả năng chịu lỗi tối ưu ($3f + 1$): Thuật toán đảm bảo tính đồng thuận nếu số lượng nút bị lỗi hoặc bị tấn công (f) không vượt quá 1/3 tổng số nút trong hệ thống. Cụ thể, để chịu được f lỗi, hệ thống cần tối thiểu $n = 3f + 1$ nút.
- Tính hoàn thiện mạnh mẽ (Strong Finality): Một khi các mục nhật ký đã được cam kết (Committed entries), chúng được coi là xác thực vĩnh viễn và không thể bị đảo ngược (Non-revertible). Điều này khác với cơ chế đồng thuận dựa trên xác suất (như Proof of Work của Bitcoin).

2 Bảng So Sánh Chi Tiết

Khía cạnh	**RAFT (CFT)**	**pBFT (BFT)**
Giả định	Các nút trung thực, chỉ gặp lỗi dừng.	Chấp nhận tối đa $1/3$ nút độc hại.
Số nút (f lỗi)	$n = 2f + 1$	$n = 3f + 1$
Độ phức tạp	Thấp (Dễ hiểu/triển khai).	Cao (Quy trình 3 giai đoạn chặt chẽ).
Thông điệp	$O(n)$mỗi vòng.	$O(n^2)$ mỗi vòng (Tốn băng thông).
Độ trễ	Thấp (1 vòng RTT).	Cao (~4 lần trễ thông điệp).
Tính hoàn thiện	Có thể bị đảo ngược (**Eventual**).	Không thể đảo ngược (**Strong**).

3 Khi Nào Sử Dụng Cái Nào?

- **Chọn RAFT khi:** Hoạt động trong môi trường tin cậy (Datacenters), mạng nội bộ ổn định, yêu cầu hiệu suất cao và không cần xử lý nút độc hại. (Ví dụ: **etcd**) .
- **Chọn pBFT khi:** Hoạt động trong môi trường không tin cậy (Blockchain), có nguy cơ nút bị xâm nhập, yêu cầu tính xác thực vĩnh viễn và chấp nhận đánh đổi tài nguyên mạng . (Ví dụ: **Hyperledger Fabric**) .

Mô tả chương trình

1. Yêu cầu & chuẩn bị môi trường (Windows)

- Python 3.11 (recommend) và pip.
- Tạo virtual environment và kích hoạt (PowerShell)

```
python -m venv .venv
.\.venv\Scripts\Activate.ps1
```

- Cài dependencies:

```
pip install grpcio grpcio-tools pytest
```

- Dọn dữ liệu/logs (nên làm trước khi chạy test durability):

```
Remove-Item -Recurse -Force .\data
Remove-Item node-*.log -Force
Remove-Item -Recurse -Force .\artifacts
New-Item -ItemType Directory -Path data,artifacts
```

(Nếu xài CMD thì chạy lệnh: rmdir /S /Q logs rmdir /S /Q artifacts mkdir logs mkdir artifacts)

2. Kiểm tra port (bắt buộc trước khi start cluster trên CI)

Project sử dụng các port gRPC mặc định 5001..5005 và HTTP status 6001..6005.

Chạy script kiểm tra port:

```
python tools/check_ports.py
```

Nếu có port bị chiếm, kill tiến trình tương ứng (Windows):

```
taskkill /PID <pid> /F
```

Ghi chú: `start_cluster.py` thực hiện preflight kiểm tra port; dùng `--force` để bỏ kiểm tra (chỉ dùng debug local).

3. Cách start node và cụm node

- Start một node trong process riêng (recommended):

```
python start_node.py 1
```

- Start toàn bộ cluster (mặc định 5 node) bằng script orchestrator (mỗi node là process riêng):

```
python start_cluster.py
# Bỏ preflight port check (debug local):
python start_cluster.py --force
```

- Chạy nhiều node trong cùng một process (phục vụ phát triển):
  python run_node.py
- Dừng tất cả node (script helper):
  python tools/stop_all_nodes.py

4. Kiểm tra trạng thái node & admin endpoints (HTTP)

Mỗi node chạy một HTTP status server tại gRPC_port + 1000. Ví dụ node gRPC 127.0.0.1:5001 -> status http://127.0.0.1:6001/state.

Các endpoint hữu dụng (được implement trong RaftNode.start_status_server):

- GET /state — trả JSON gồm: role, leader_id, term, log_len, commit_index, last_applied, blackholed_peers, kv_snapshot, next_index, match_index, replication_errors.
- GET /admin/disconnect?peers=ID1,ID2 — node sẽ thêm peer vào blackholed_peers và bỏ qua replicate tới peer đó.
- GET /admin/reconnect?peers=ID1,ID2 — loại peer khỏi blackholed_peers.
- GET /admin/clear — xóa mọi blackhole.
- GET /admin/shutdown — tắt node (gọi graceful stop).
- GET /admin/setterm?term=NN — đặt current_term (bị giới hạn: từ chối giá trị lớn quá >=1000).

Ví dụ PowerShell:

```
Invoke-RestMethod "http://127.0.0.1:6001/state"
Invoke-RestMethod "http://127.0.0.1:6001/admin/disconnect?peers=2,3"
Invoke-RestMethod "http://127.0.0.1:6001/admin/reconnect?peers=2,3"
Invoke-RestMethod "http://127.0.0.1:6001/admin/shutdown"
```

5. Gửi lệnh từ client

- Dùng client CLI wrapper:

```
python raft_client.py set mykey 123
# hoặc (hỗ trợ tên file cũ):
python raft_cilent.py set mykey 123
```

- Hàm chủ chốt gửi lệnh: raft_client.send_command(command, max_attempts=3, backoff=0.5).
- send_command thực hiện find_leader() (dùng /state) rồi gọi ClientAppend RPC (fallback AppendEntries nếu server trả UNIMPLEMENTED).

6. Cách thay đổi số node / ports / topology

- Mở file cấu hình: config.py
- Sửa NODES (dict) theo định dạng node_id: "host:port" (ví dụ thêm 6: "127.0.0.1:5006").
- NUM_NODES, ALL_NODES và MAJORITY được tính tự động từ NODES.
- Sau khi chỉnh NODES:
    - Đảm bảo các port mới không trùng (dùng tools/check_ports.py).
    - Restart toàn cluster (kill tiến trình cũ, sau đó python start_cluster.py).

Lưu ý: code hiện tại giả định các node id là liên tiếp 1..N ở nhiều script; nếu thay đổi phức tạp hơn (ví dụ id không liên tiếp), kiểm tra `start_cluster.py` và `tests` để đảm bảo tương thích.

7. Mô phỏng lỗi / nút độc hại (Byzantine)

- Partition / blackhole: dùng /admin/disconnect trên node A để khiến node A bỏ replicate tới một số peer. Việc này mô phỏng mất kết nối một chiều.
- Shutdown node: /admin/shutdown hoặc kill PID.
- Ép term để kích thích election: /admin/setterm?term=NN.
- pBFT demo (mô phỏng Byzantine node):
- File demo: run_pbft_node.py (tạo PBFTNode với byzantine=(i == 3) trong ví dụ).
- Để thay node gian lận, sửa run_pbft_node.py hoặc pbft_node.py.

8. Persistence & durability

- KV store file-backed: mỗi node lưu dữ liệu ở data/node_<ID>.json bằng class KVStore (file: kv_store.py).
- Hàm: KVStore.set(key, value) và KVStore.get(key).
- Durability test (kịch bản test sẵn có): tests/test_durability.py
- Mô tả: start cluster, gửi set dur_key 42, kill PIDs, restart cluster, kiểm tra dur_key tồn tại trong kv_snapshot trả bởi /state.
- Chạy bằng:

```
python -m pytest tests/test_durability.py::test_durability -q
```

9. File log & artifacts

- Logs của node khi start bằng start_node.py: node-1.log, node-2.log, ...
- Khi test thất bại, tests/test_durability.py có helper dump_logs() để copy logs vào artifacts/<timestamp>_reason/.

10. Các bước chạy cụ thể (step-by-step)

1. Mở PowerShell, di chuyển vào thư mục project.
2. Tạo và kích hoạt venv (như ở mục 1).
3. (CI) Chạy python tools/check_ports.py để đảm bảo các port 5001..5005 và 6001..6005 trống.
4. Khởi cụm 5 node:

   python start_cluster.py

   Hoặc start 1 node để debug:

   python start_node.py 1
5. Xác minh trạng thái node (ví dụ node 1):

   Invoke-RestMethod "http://127.0.0.1:6001/state"
6. Gửi lệnh ví dụ:

   python raft_client.py set example 100
7. Kiểm tra kv_snapshot trong /state của các node để xác nhận commit.
8. Mô phỏng fault: tắt leader bằng /admin/shutdown hoặc blackhole follower bằng /admin/disconnect.
9. Chạy test tổng quát, chạy test durability và pBFT:
   - 9.1. Mở terminal (Powershell/CMD) mới và cd vào thư mục chứa đồ án
   - 9.2. Kill toàn bộ python và port còn dư: taskkill /F /IM python.exe
   - 9.3. Dọn dữ liệu/logs:

     Remove-Item -Recurse -Force .\data
     Remove-Item node-*.log -Force
     Remove-Item -Recurse -Force .\artifacts
     New-Item -ItemType Directory -Path data,artifacts

     (Nếu xài CMD thì chạy lệnh: rmdir /S /Q logs rmdir /S /Q artifacts mkdir logs mkdir artifacts)

   - 9.4. Kích hoạt môi trường: .venv\Scripts\activate
   - 9.5. Chạy FULL TEST python -m pytest -q
   - 9.6. Kill python lần nữa: taskkill /F /IM python.exe
   - 9.7. Dọn sạch lại dữ liệu/logs:

     Remove-Item -Recurse -Force .\data
     Remove-Item node-*.log -Force
     Remove-Item -Recurse -Force .\artifacts
     New-Item -ItemType Directory -Path data,artifacts

(Nếu xài CMD thì chạy lệnh: rmdir /S /Q logs rmdir /S /Q artifacts mkdir logs mkdir artifacts)

9.8. Test riêng DURABILITY

```
python -m pytest tests/test_durability.py::test_durability -q
```

9.9. Test pBFT:

```
python start_pbft_cluster.py

python -m pytest -q tests/test_pbft.py
```

10. Khi test thất bại, kiểm tra artifacts/ và node-*.log để phân tích.

Tự đánh giá

1. Ưu điểm chính

- Kiến trúc rõ ràng: tách biệt RaftState (logic state) và RaftNode (gRPC, replication, admin) giúp dễ hiểu và bảo trì.
- Hỗ trợ E2E: có script start_node.py / start_cluster.py để chạy multi-process, thuận tiện cho mô phỏng cluster thực tế.
- Công cụ debug / admin mạnh: HTTP status + admin endpoints (/state, /admin/disconnect, /admin/shutdown, /admin/setterm) giúp inject lỗi và thu thông tin trạng thái nhanh.
- Durability thực hiện hợp lý: KVStore dùng temp-file + replace để đảm bảo atomic persist giữa các node.
- Replication giàu tính năng: replicate_to_peer() áp dụng fast-path + probing + backoff, và xử lý higher-term detection.
- Test tích hợp: có tests/test_durability.py và helpers trong tools/ cho kịch bản start→commit→kill→restart→verify.

2. Nhược điểm chính

- Script/tests hard-coded cho 5 node và ports -> khó thay topology.
- Quản lý tiến trình/port fragile; tests có thể phá runner nếu môi trường bận.
- Logging dùng print, thiếu cấu hình và định dạng có cấu trúc.
- Thiếu xử lý tín hiệu/graceful shutdown; admin HTTP không có xác thực.

3. Giải pháp khắc phục (ý tưởng)

- Làm cho scripts và tests hoạt động động theo config.NODES để hỗ trợ số node/port tuỳ ý.
- Thiết lập logging chuẩn (logging) với cấu hình qua biến môi trường hoặc CLI và xuất log dạng có cấu trúc để dễ phân tích.
- Cải thiện quản lý tiến trình và lifecycle: orchestrator trả PIDs/trạng thái, tests dùng retry/backoff, và thêm xử lý tín hiệu để graceful shutdown.
- Bảo vệ admin HTTP (token/whitelist tùy chọn) và cập nhật tài liệu hướng dẫn cấu hình/ports.

Thông tin thành viên và phân chia công việc

No	MSSV	Họ và tên	Gmail	Công việc
1	22127210	Phạm Anh Khôi	pakhoi22@clc.fitus.edu.vn	Tìm hiểu thuật toán Viết report
2	22127092	Lê Bảo Giang	lbgiang22@clc.fitus.edu.vn	Cài đặt pBFT
3	20127656	Trần Minh Trường	tmtruong20@clc.fitus.edu.vn	Mô tả chương trình Viết report
4	22127133	Đinh Vũ Huân	dinhvuhuan282004@gmail.com	Cài đặt RAFT
5	21127594	Phạm Huỳnh Tấn Đạt	phtdat21@clc.fitus.edu.vn	Clean code Cài đặt pBFT

Reference

1. Practical Byzantine Fault Tolerance
2. In Search of an Understandable Consensus Algorithm