Laporan Tugas Individu 2

Sistem Paralel dan Terdistribusi A

Disusun Oleh :

Rafi Baydar Athaillah 11231081

10 Oktober 2025

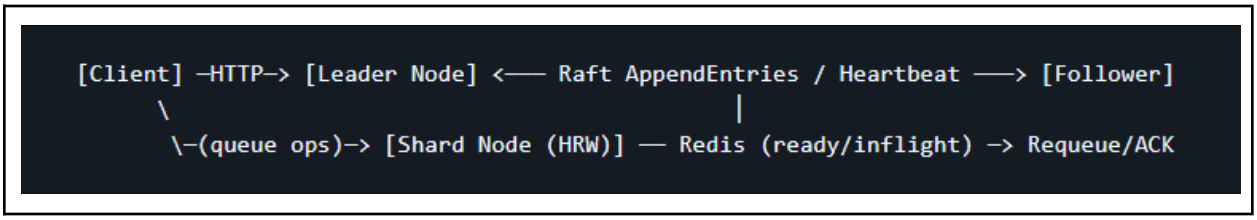
Arsitektur Sistem

Sistem terdiri dari tiga node aplikasi (node-1, node-2, node-3) yang mengekspos HTTP API pada port 8080/8081/8082 serta satu Redis (port 6379) sebagai penyimpanan antrian dan cache metadata tertentu. Ketiga node membentuk cluster Raft untuk leader election dan replicated log, sehingga perubahan status global—khususnya state Distributed Lock Manager (DLM)—hanya di-commit setelah mencapai mayoritas (quorum). Dengan demikian, operasi yang memengaruhi lock state (acquire/release) bersifat linearizable melalui leader, sementara jalur distributed queue memanfaatkan Redis untuk at-least-once delivery via visibility timeout (+ requeue otomatis).

Komponen Utama

1. **Raft Consensus**
   Tiap node dapat berperan sebagai *Follower/Candidate/Leader*. Leader menerima command (misal *LOCK ACQ/REL*), menuliskannya ke log, mereplikasi via *AppendEntries*, lalu menandai commit setelah mendapat mayoritas; barulah efek diterapkan ke state machine (tabel holders, waitq, dll.).
2. **Distributed Lock Manager (DLM)**
   Mendukung mode *shared (s)* dan *exclusive (x)*. Acquire akan grant bila tidak konflik; jika konflik, permintaan dimasukkan ke wait queue dan wait-for graph diupdate untuk keperluan deadlock detection (DFS siklus). Release akan membangunkan batch shared atau satu exclusive sesuai prioritas.
3. **Distributed Queue**
   Partisi topik menggunakan *Rendezvous/HRW hashing* sehingga topic dipetakan konsisten ke shard (node) tertentu. Produksi pesan menulis ke ready list di Redis; consume memindahkan pesan ke inflight beserta *visible_until*. Jika tidak di-ACK sebelum TTL habis, pesan akan requeue otomatis. Idempotency diterapkan via *msg_id = md5(payload_normalized)*.
4. **Cache (MESI-lite)**
   Tiap node menyimpan entry lokal dengan LRU; saat ada *PUT/write-through*, node melakukan *invalidate broadcast* ke *peers (state I)* melalui RPC internal untuk menjaga koherensi sederhana tanpa *directory* kompleks.
5. **Failure Detector**
   Tiap node melakukan *ping* periodik ke */health* setiap *peer* untuk mencatat *UP/DOWN* dan *RTT*; sinyal ini dipakai untuk diagnosa dan *routing awareness*.

Topologi dan Komunikasi



- **Client → Leader**: semua mutasi lock diarahkan ke leader. *Client/benchmark* mendeteksi *409 not_leader* dan *follow* ke *alamat leader (hint)* atau *probe /raft/health*.
- **Client → Shard (Queue)**: operasi queue (produce/consume/ack) dapat langsung menuju node hasil HRW hashing atas topic, sementara data *ready/inflight* disimpan di Redis agar requeue berbasis *visibility TTL* berjalan deterministik.
- **Inter-node**: protokol *Raft* (replication/commit) dan invalidate RPC (cache).
- **Observability**: */health*, */raft/health*, *failure detector snapshot*, serta *OpenAPI* untuk kontrak antarmuka.

Algoritme Yang Digunakan

1. Raft Consensus (ringkas)

- Roles: Follower → Candidate → Leader.
- Timers: election timeout acak per node; leader mengirim heartbeat.
- Log replication: leader menerima propose(cmd) → append ke log → kirim AppendEntries → commit berdasar mayoritas → apply_fn(cmd) (Lock ACQ/REL).
- Safety: semua side-effect (lock state) hanya terjadi setelah entry committed.

2. Distributed Lock Manager

- Mode: shared (s) dan exclusive (x).
- Struktur: holders, modes, waitq, waits_for.
- Acquire:
  - kosong → grant; shared jika mode current shared → grant; konflik → antri di waitq + bangun relasi waits_for.
- Release: kosongkan holder; jika kosong → grant batch untuk shared berturut-turut atau satu exclusive.
- Deadlock detection: DFS pada wait-for graph (warna white/gray/black), deteksi siklus dan laporkan siklus pemilik.

3. Distributed Queue (HRW + Redis)

- Partitioning: Rendezvous/HRW hashing score = md5(key|node) pilih node skor tertinggi → topic konsisten ke shard.
- Ready/Inflight: ready = list, inflight = hash{visible_until -> messageJson}.
- At-least-once: consume memindah ke inflight dengan TTL; jika tak di-ACK → requeue.
- Idempotency: msg_id = md5(json(payload, sort_keys=True)).

4. Cache Coherence (MESI-lite)

- State: M (modified, hanya owner), E (exclusive clean), S (shared), I (invalid).
- Implementasi ringkas: invalidate broadcast saat PUT (write-through) → peers set I (tanpa directory kompleks). Lokalisasi objek cache per-node dengan LRU (max items).

5. Failure Detector

- Ping periodik ke /health tiap peer → status UP/DOWN dan RTT; event callback untuk logging dan (opsional) routing awareness.

API Documentation (OpenAPI)

Endpoint

1. Health & Raft:
   - GET /health → status node
   - GET /raft/health → state (leader|follower), term
   - GET /fd → snapshot failure detector
2. Lock:
   - POST /lock/acquire {resource, owner, mode:'s|x'}
   - POST /lock/release {resource, owner}
3. Queue:
   - POST /queue/produce {topic, payload}
   - GET /queue/consume?topic=...&max=N&visibility_ttl=SEC
   - POST /queue/ack {topic, msg_id}
4. Cache:
   - GET /cache/{key}
   - PUT /cache/{key} (body: arbitrary JSON)
5. RPC internal:
   - POST /rpc (JSON-RPC; cache.invalidate)

Contoh Request/Response

- Lock Acquire

```
POST /lock/acquire
Content-Type: application/json

{"resource":"res-A","owner":"cli-1","mode":"x"}
```

```
{"status":"enqueued_or_granted"}
```

- Queue Procedure

```
POST /queue/produce
{"topic":"orders","payload":{"id":1,"ts":"2025-10-28T03:00:00Z"}}
```

```
{"produced":true,"msg_id":"f37e51db6b4cff69d1635782d18819d2"}
```

OpenAPI (YAML)

```
openapi: 3.0.3
info: { title: Distributed Sync System API, version: "1.0.0" }
servers: [{ url: http://localhost:8080 }]
paths:
  /health:
    get: { summary: Health, responses: { '200': { description: OK } } }
  /lock/acquire:
    post:
      summary: Acquire lock (s|x)
      requestBody:
        required: true
        content: { application/json: { schema:
          { type: object, properties:
            { resource: {type: string}, owner: {type: string}, mode: {type: string, enum: [s,x]} } } } }
      responses: { '200': { description: ok } }
  /lock/release:
    post:
      summary: Release lock
      requestBody:
        required: true
        content: { application/json: { schema:
          { type: object, properties: { resource: {type: string}, owner: {type: string} } } } }
      responses: { '200': { description: ok } }
  /queue/produce:
    post: { summary: Produce, responses: { '200': { description: ok } } }
  /queue/consume:
    get:
      summary: Consume
      parameters:
        - in: query; name: topic; schema: {type: string}
        - in: query; name: max; schema: {type: integer, default: 10}
        - in: query; name: visibility_ttl; schema: {type: integer, default: 30}
      responses: { '200': { description: ok } }
  /queue/ack:
    post: { summary: Ack, responses: { '200': { description: ok } } }
  /cache/{key}:
    get: { summary: Get, parameters: [{in: path, name: key, required: true, schema: {type: string}}], responses: {'200': {description: ok}} }
    put: { summary: Put, parameters: [{in: path, name: key, required: true, schema: {type: string}}], responses: {'200': {description: ok}} }
```

Deployment Guide & Troubleshooting

1. Prasyarat
    - Docker & Docker Compose.
    - Windows PowerShell.
    - Port 8080/8081/8082 terbuka; Redis di 6379.

2. Langkah Deploy

```
docker compose -f docker\docker-compose.yml down -v

docker compose -f docker\docker-compose.yml up -d --build

$ports = 8080,8081,8082
foreach($p in $ports){ try{ Invoke-RestMethod "http://localhost:$p/health" }catch{} }
function Get-Leader {
  foreach($p in 8080,8081,8082){ try{ $r=Invoke-RestMethod "http://localhost:$p/raft/health"; if($r.state -eq 'leader'){ return "http://localhost:$p" }
}catch{} }
  return $null
}
$L = Get-Leader
```

3. Quick Test

```
# Lock
$acq = @{ resource='res-A'; owner='cli-1'; mode='x' } | ConvertTo-Json -Compress
Invoke-RestMethod -Method POST "$L/lock/acquire" -ContentType 'application/json' -Body $acq
$rel = @{ resource='res-A'; owner='cli-1' } | ConvertTo-Json -Compress
Invoke-RestMethod -Method POST "$L/lock/release" -ContentType 'application/json' -Body $rel

# Queue
$b = @{ topic='orders'; payload=@{ id=1; ts=[DateTime]::UtcNow } } | ConvertTo-Json -Compress
Invoke-RestMethod -Method POST "$L/queue/produce" -ContentType 'application/json' -Body $b
$c = Invoke-RestMethod "$L/queue/consume?topic=orders&max=1&visibility_ttl=30"
$ack = @{ topic='orders'; msg_id=$c.messages[0].msg_id } | ConvertTo-Json -Compress
Invoke-RestMethod -Method POST "$L/queue/ack" -ContentType 'application/json' -Body $ack
```

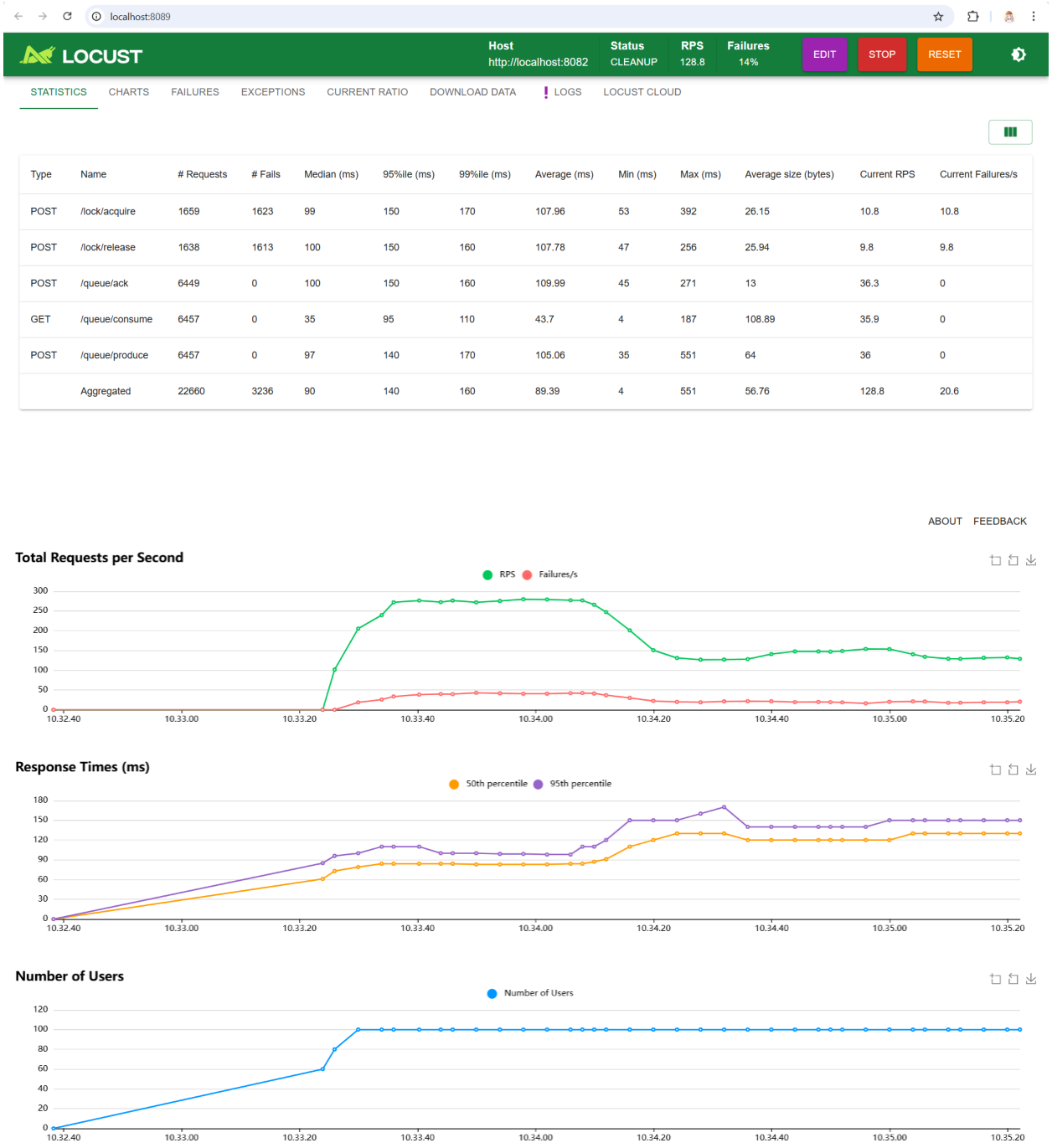
4. Troubleshooting
    - ***{"error":"not_leader"}*** = Panggil *Get-Leader* dan ulangi request ke leader.
    - ***JSONDecodeError* saat pakai *curl.exe*** = Gunakan *Invoke-RestMethod* + *ConvertTo-Json -Compress* (PowerShell lebih aman untuk quoting).
    - **Leader tidak terdeteksi** = tunggu beberapa detik (election), cek *docker compose logs -f node1 node2 node3*.
    - **Pesan tidak ter-requeue** = pastikan *requeue_daemon* aktif (lihat logs), cek *visibility_ttl* cukup pendek saat uji.

Performance Analysis Report

Benchmarking hasil dengan berbagai skenario

```
(.venv) PS E:\.Tugas ITK\SISTER (Sistem Paralel dan Terdistribusi)\TUGAS 2> python -m benchmarks.load_test_scenario `
>>   --scenario lock `
>>   --leader "$L" `
>>   --nodes "$NODES" `
>>   --iterations 1000 `
>>   --concurrency 50 `
>>   --timeout 5 `
>>   --grant-wait-ms 0 `
>>   --verbose
------------------------------------------------------------------------
[INFO] Starting benchmark...
[INFO] Target URL   : http://localhost:8082/lock/*
[INFO] Total Requests (approx) : 2000
[INFO] Concurrency  : 50
------------------------------------------------------------------------
------------------------------------------------------------------------
Benchmark Results:
------------------------------------------------------------------------
Total Requests       :   2000
Successful Requests  :   2000
Failed Requests      :      0
Average Latency      :    80.39 ms
Total Time Taken     :     3.28 s
Throughput           :   610.25 req/sec
------------------------------------------------------------------------
Per-operation breakdown:
Op                     Req       OK    Avg(ms)
lock.acquire          1000     1000      80.64
lock.release          1000     1000      80.14
------------------------------------------------------------------------
[INFO] Scenario      : lock
[INFO] Target        : http://localhost:8082/lock/*
[INFO] Concurrency   : 50
[INFO] Iterations    : 1000
========================================================================
(.venv) PS E:\.Tugas ITK\SISTER (Sistem Paralel dan Terdistribusi)\TUGAS 2>
```

Hasil Pengujian dengan Locust



Type	Name	# Requests	# Fails	Median (ms)	95%ile (ms)	99%ile (ms)	Average (ms)	Min (ms)	Max (ms)	Average size (bytes)	Current RPS	Current Failures/s
POST	/lock/acquire	1659	1623	99	150	170	107.96	53	392	26.15	10.8	10.8
POST	/lock/release	1638	1613	100	150	160	107.78	47	256	25.94	9.8	9.8
POST	/queue/ack	6449	0	100	150	160	109.99	45	271	13	36.3	0
GET	/queue/consume	6457	0	35	95	110	43.7	4	187	108.89	35.9	0
POST	/queue/produce	6457	0	97	140	170	105.06	35	551	64	36	0
	Aggregated	22660	3236	90	140	160	89.39	4	551	56.76	128.8	20.6