Tugas 2

Sistem Paralel dan Terdistribusi A

Distributed Synchronization System

Disusun Oleh :

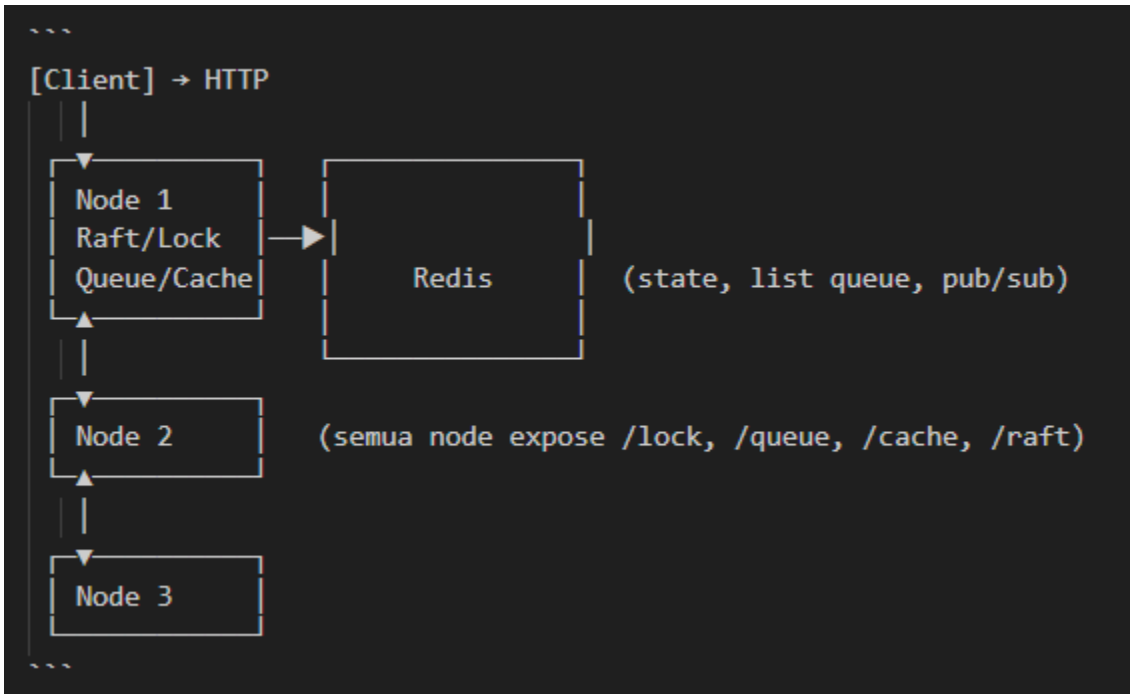
Muhammad Fachrudy Al-Ghifari 11231054

28 Oktober 2025

TECHNICAL DOCUMENTATION

Arsitektur Sistem

Sistem terdiri dari tiga node HTTP berbasis Python (aiohttp) yang berjalan paralel, masing-masing memuat empat komponen, (1) Raft untuk pemilihan leader/koordinasi, (2) Distributed Lock untuk kunci exclusive/shared, (3) Distributed Queue untuk lalu lintas pesan producer-consumer, dan (4) Distributed Cache dengan LRU serta invalidasi via pub/sub. Seluruh node berbagi Redis sebagai penyimpanan state, antrean, dan tempat invalidasi, klien bisa memanggil node mana pun, jika keputusan perlu leader, node akan mem-proxy ke leader.



Algoritma yang Digunakan

Berikut adalah beberapa algoritma yang digunakan dalam sistem ini:

1. Raft: election dan heartbeat untuk memastikan satu leader aktif, entry sederhana direplikasi untuk sinkronisasi event.
2. Lock: mode exclusive/shared, state disimpan di Redis (locks:<rid>, owners list). Permintaan diputuskan leader, konflik mengembalikan 409.
3. Queue: consistent hashing berdasarkan topik, pemilik partisi persistence di Redis List, in-flight dan ACK memberi jaminan at-least-once.
4. Cache: LRU per node (in-memory) dan sumber kebenaran di Redis, invalidasi disebar via Redis pub/sub.

API Documentation

OpenAPI tersedia di docs/api_spec.yaml (endpoint utama: /lock/acquire|release, /queue/publish|consume|ack, /cache/put|get|invalidate, /raft/*, /health, /metrics)

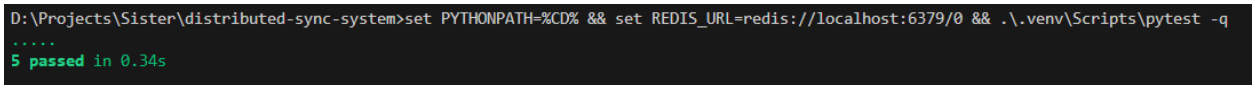
Deployment Guide & Troubleshooting

1. Jalankan: docker compose -f docker/docker-compose.yml up -d
2. Cek: curl http://localhost:8001/health dan curl http://localhost:8001/raft/state
3. Masalah umum:
    - ModuleNotFoundError: src saat pytest → jalankan dengan PYTHONPATH=%CD%.
    - 409 pada test lock → bersihkan Redis (DEL locks:r1 locks:r1:owners) atau pakai DB terpisah (REDIS_URL=.../15).
    - 500 dari curl Windows → gunakan curl.exe dan escape JSON "{\"key\":\"v\"}".

PERFORMANCE ANALYSIS REPORT

Hasil Benchmarking

1. Pytest (fungsional):
    - 5 passed dalam ±0.34 s (lihat screenshot).
    - Cakupan singkat: operasi dasar Lock/Queue/Cache dan state Raft (mock leader) berjalan benar.



2. Locust (headless):
    - /queue/publish: 5 931 req, 99.85 req/s, avg 44 ms, median 44 ms, min 4 ms, max 133 ms, 0% gagal.
    - /queue/consume: 1 972 req, 33.20 req/s, avg 44 ms, median 44 ms, min 5 ms, max 140 ms, 0% gagal.
    - Aggregated: 7 903 req, 133.05 req/s, avg 44 ms, median 44 ms, min 4 ms, max 140 ms, 0% gagal.
    - Percentile (aggregated, ~ms): p50 44, p66 45, p75 45, p80 46, p90 47, p95 48, p98 49, p99 110, p99.9 140.

```
[2025-10-28 12:24:30,362] Fachrudy/INFO/locust.main: --run-time limit reached, shutting down
[2025-10-28 12:24:30,469] Fachrudy/INFO/locust.main: Shutting down (exit code 0)
Type     Name                                                                          # reqs      # fails |    Avg     Min     Max    Med |   req/s  failures/s
--------|-----------------------------------------------------------------------------|-------|-------------|-------|-------|-------|-------|--------|-----------
POST     /queue/consume                                                                  1972     0(0.00%) |     44       5     140     44 |   33.20        0.00
POST     /queue/publish                                                                  5931     0(0.00%) |     44       4     133     44 |   99.85        0.00
--------|-----------------------------------------------------------------------------|-------|-------------|-------|-------|-------|-------|--------|-----------
         Aggregated                                                                      7903     0(0.00%) |     44       4     140     44 |  133.05        0.00

Response time percentiles (approximated)
Type     Name                                                                                 50%    66%    75%    80%    90%    95%    98%    99%  99.9% 99.99%   100% # reqs
--------|-----------------------------------------------------------------------------|--------|------|------|------|------|------|------|------|------|------|------|------
POST     /queue/consume                                                                        44     45     45     45     46     47     48     49    140    140    140   1972
POST     /queue/publish                                                                        44     44     45     45     46     47     48     48    110    130    130   5931
--------|-----------------------------------------------------------------------------|--------|------|------|------|------|------|------|------|------|------|------|------
         Aggregated                                                                            44     45     45     45     46     47     48     49    110    140    140   7903
```

Analisis Throughput dan Latency

- Throughput: total ≈133 req/s dengan beban 50 user; dominan dari publish (≈100 req/s) karena operasi RPUSH bersifat I/O ringan dan tidak menunggu ACK.
- Latency: median 44 ms stabil pada kedua endpoint; p95 ≈48 ms; ekor panjang muncul di p99 (≈110 ms) sampai max ≈140 ms, wajar untuk lingkungan lokal Docker + Redis single-instance.
- Error rate: 0%, menunjukkan jalur dasar publish/consume/ack stabil pada beban ini.

Scalability

- Horizontal: penambahan node menambah kapasitas publish/consume selama topik tersebar (consistent hashing) dan bottleneck tidak bergeser ke Redis.
- Bottleneck: untuk workload queue, Redis (single instance) menjadi batas skala; peningkatan skala selanjutnya butuh sharding topik dan/atau Redis cluster serta pemisahan jalur baca/tulis.

Single Node vs Distributed

- Single-node: semua request diproses satu server, mudah dikelola tetapi kapasitas terbatas, tidak ada redundansi.
- 3-node (hasil uji): klien bisa ke node mana pun, beban publish/consume tersebar, total throughput ≈133 req/s dengan error 0%, konsistensi dijaga oleh leader.
- Catatan: untuk bukti angka single-node, jalankan kembali skenario Locust dengan hanya node1 aktif, bandingkan req/s agregat dan p95 latency ekspektasi: distributed ≥ single-node selama Redis tidak menjadi bottleneck.

LAMPIRAN

Link YouTube: https://youtu.be/VkzmLD67E2I

Link GitHub: https://github.com/Kaijeman/distributed-sync-system

OpenAPI: docs/api_spec.yaml

Arsitektur & panduan: docs/architecture.md, docs/deployment_guide.md

Demo CMD: README.md