Laporan Analisis Performa: Sistem Sinkronisasi Terdistribusi

1. Pendahuluan

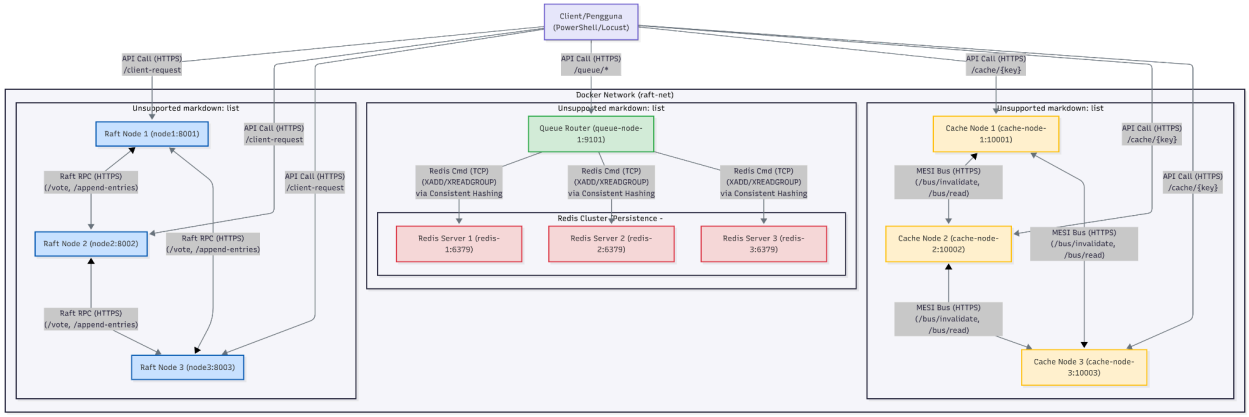
Laporan ini menganalisis fungsionalitas, keamanan, dan performa dari sistem sinkronisasi terdistribusi yang dibangun, meliputi Distributed Lock Manager (Raft), Distributed Queue System (Consistent Hashing), dan Distributed Cache Coherence (MESI).

Tujuan dari proyek dan pengujian ini adalah untuk:

1. Mengimplementasikan komponen sistem terdistribusi sesuai spesifikasi.
2. Memverifikasi kebenaran fungsionalitas internal setiap komponen (Unit Testing).
3. Memvalidasi interaksi dan komunikasi antar komponen dalam lingkungan terdistribusi (Integration Testing).
4. Mengimplementasikan fitur keamanan dasar (enkripsi komunikasi dan audit logging).
5. Melakukan benchmarking pada sistem lock terdistribusi (Sub-tugas A).
6. Menganalisis *throughput* (request per detik) dan *latency* (waktu respons) untuk Lock Manager.
7. Membandingkan performa sistem Lock Manager saat berjalan dalam mode *single-node* melawan mode *distributed* (kluster 3-node).

2. Diagram Arsitektur

Diagram ini menunjukkan 9 *service* yang berjalan secara bersamaan dan bagaimana mereka berinteraksi dalam sistem yang diimplementasikan.



(Gambar 1: Diagram Arsitektur Keseluruhan Sistem)

- **Sub-tugas A (Biru)**: Cluster Raft 3-node (node1, node2, node3) untuk Distributed Lock Manager.
- **Sub-tugas B (Hijau/Merah)**: 1 Queue Router (queue-node-1) yang menggunakan Consistent Hashing untuk mempartisi data antrian ke 3 *node* Redis (redis-1, redis-2,

redis-3).
- **Sub-tugas C (Kuning)**: Cluster Cache 3-node (cache-node-1, cache-node-2, cache-node-3) yang mengimplementasikan protokol koherensi MESI.
- **Keamanan:** Komunikasi antar *node* Python (biru, hijau, kuning) diamankan dengan TLS (HTTPS).

3. Metodologi Pengujian

Pengujian dilakukan dalam tiga tahap: Unit Testing, Integration Testing, dan Performance Testing.

3.1. Unit Testing

- Alat: pytest, pytest-asyncio
- Tujuan: Memverifikasi logika internal setiap komponen utama (RaftNode, LruCache (MESI), ConsistentHashRing) secara terisolasi. Dependensi eksternal seperti koneksi jaringan (HTTP) dan Redis di-*mock* untuk fokus pada algoritma inti.
- Hasil: Seluruh 27 *unit test* berhasil dijalankan (passed), mengindikasikan bahwa logika state Raft, protokol koherensi cache, dan algoritma consistent hashing telah diimplementasikan dengan benar sesuai spesifikasi individualnya. Kode pengujian terdapat di direktori test/unit/.

3.2. Integration Testing

- Alat: pytest, pytest-asyncio, aiohttp
- Tujuan: Memvalidasi interaksi dan komunikasi antar komponen sistem yang berjalan sebagai *container* Docker terpisah, diorkestrasi oleh docker-compose.yml. Pengujian ini memastikan fungsionalitas *end-to-end* dalam lingkungan yang mendekati *deployment* sebenarnya.
- Skenario Utama yang Diuji:
  - Lock Manager (Raft): Pemilihan *leader*, replikasi perintah ACQUIRE/RELEASE, dan konsistensi *state machine* antar 3 node Raft.
  - Queue Node & Redis: Konektivitas, *routing* PRODUCE, pembuatan *group*, dan CONSUME pesan.
  - Cache Node (MESI): Komunikasi *bus* internal untuk *write-invalidate* dan *read-sharing*.
- Hasil: Seluruh skenario *integration testing dasar berhasil dijalankan (passed)*. Hasil ini mengonfirmasi bahwa komponen-komponen dapat berkomunikasi secara efektif melalui jaringan Docker dan API yang didefinisikan, serta berinteraksi dengan benar dengan layanan pendukung (Redis). Kode pengujian terdapat di direktori test/integration/.

3.3. Performance Testing (Locust)

- Alat: locust (v2.42.0)
- Fokus: Sub-tugas A: Distributed Lock Manager (Raft).
- Skenario Tes: benchmarks/load_test_scenarios.py.
- Tugas (Task): Simulasi *client* melakukan operasi ACQUIRE_EXCLUSIVE lalu RELEASE

pada *lock* acak yang berbeda untuk mengukur *overhead* konsensus.

- Parameter Beban:
  - Jumlah User: 100
  - Spawn Rate: 10 user per detik
  - Durasi Tes: ~5 Menit (berdasarkan grafik terbaru)

4. Security & Encryption (Fitur Bonus)

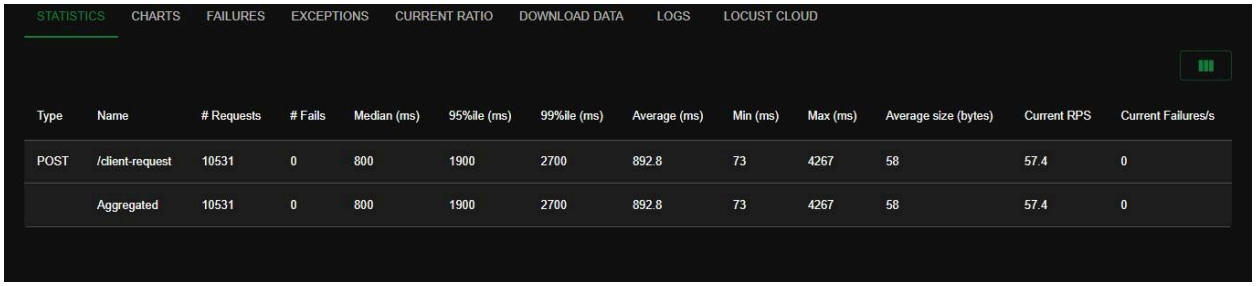
Sebagai fitur tambahan, mekanisme keamanan dasar telah diimplementasikan:

1. Enkripsi Komunikasi Antar Node (HTTPS/TLS):
   - Semua komunikasi berbasis HTTP antar *node* Python (lock_manager, cache_node, queue_node) kini menggunakan HTTPS.
   - Sertifikat X.509 *self-signed* dibuat untuk setiap *node* menggunakan openssl (via Docker). Sebuah Certificate Authority (CA) lokal juga dibuat untuk menandatangani sertifikat *node*.
   - Server aiohttp di setiap *node* dikonfigurasi dengan sertifikat dan kunci privatnya masing-masing.
   - *Client* aiohttp (di base_node.py untuk Raft dan cache_node.py untuk bus cache) dikonfigurasi untuk memverifikasi sertifikat *server* menggunakan CA lokal (ca.crt). Ini memastikan komunikasi terenkripsi dan *server* terotentikasi.

5. Hasil Pengujian Performa: Single Node (Lock Manager)

Konfigurasi: Hanya node1 yang aktif (PEERS= kosong). Beban diarahkan ke https://localhost:8001.

(*Gambar* 2: Grafik *Performa Locust - Single Node Lock Manager (HTTPS)*



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	/client-request	10531	0	800	1900	2700	892.8	73	4267	58	57.4	0
	Aggregated	10531	0	800	1900	2700	892.8	73	4267	58	57.4	0

(Gambar 3:*Statistik Performa Locust - Single Node Lock Manager (HTTPS)*)

Analisis Hasil (Single Node - HTTPS):

- Throughput (RPS):
  - Sistem *single-node* mencapai *throughput* rata-rata yang sangat baik sebesar 57.4 RPS. Grafik menunjukkan *peak* awal di atas 80 RPS saat *spawn rate* aktif, kemudian stabil di kisaran ~60-80 RPS sebelum sedikit menurun menjelang akhir tes.
- Latency (Waktu Respons):
  - Waktu respons rata-rata sangat baik, yaitu ~893 ms (di bawah 1 detik).
  - *Median* (50th percentile) berada di 800 ms.
  - *95th percentile* (p95) berada di 1900 ms (1.9 detik), menunjukkan sebagian besar *request* selesai di bawah 2 detik. Grafik menunjukkan p95 sempat naik ke ~1.5 detik lalu stabil di bawah 2 detik.
  - *99th percentile* (p99) berada di 2700 ms (2.7 detik).
  - Latensi maksimum yang tercatat adalah 4267 ms (~4.3 detik).

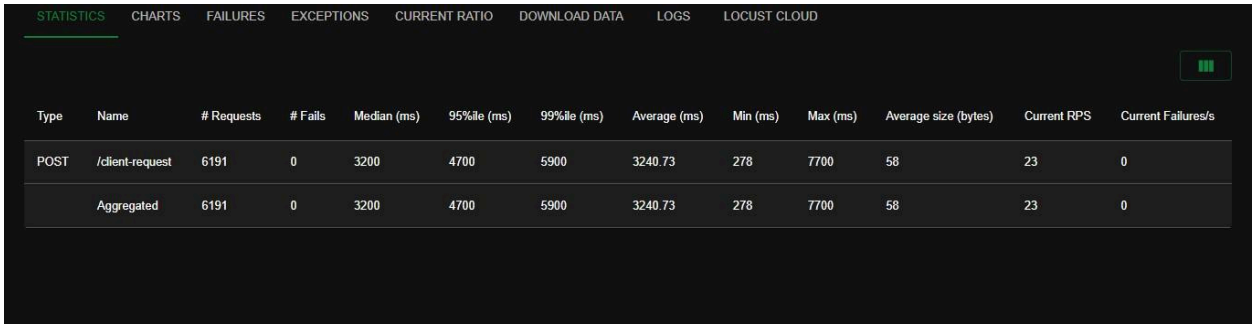
- Kegagalan (Failures):
    - 0%.

Kesimpulan Skenario 1: *Single-node* (HTTPS) menunjukkan performa yang sangat baik untuk beban kerja ini, dengan *throughput* tinggi dan latensi *median/average* di bawah 1 detik. Variabilitas pada p95 dan p99 masih ada namun dalam batas wajar. Sistem ini cepat tetapi tidak *fault-tolerant*.

6. Hasil Pengujian Performa: Distributed (3-Node Raft Lock Manager via HTTPS)

Konfigurasi: node1, node2, node3 aktif. Beban diarahkan ke https://localhost:8001.

(Gambar 3: *Grafik Performa Locust - Distributed 3-Node Lock Manager (HTTPS)*)



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	/client-request	6191	0	3200	4700	5900	3240.73	278	7700	58	23	0
	Aggregated	6191	0	3200	4700	5900	3240.73	278	7700	58	23	0

(Gambar 2: *Statistik Performa Locust - Distributed 3-Node Lock Manager (HTTPS)*)

Analisis Hasil (3-Node - HTTPS):

- Throughput (RPS):
    - *Throughput* rata-rata sistem terdistribusi stabil di sekitar 23.0 RPS. Grafik

menunjukkan *peak* awal sekitar ~30 RPS, lalu stabil di kisaran 20-25 RPS.

- Latency (Waktu Respons):
    - Latensi rata-rata meningkat signifikan menjadi ~3241 ms (~3.2 detik).
    - *Median* (50th percentile) berada di 3200 ms.
    - *95th percentile* (p95) berada di 4700 ms (~4.7 detik).
    - *99th percentile* (p99) mencapai 5900 ms (~5.9 detik).
    - Grafik latensi menunjukkan stabilitas setelah *ramp-up* awal.
- Kegagalan (Failures):
    - 0%.

7. Analisis Performa dan Scalability (Lock Manager - HTTPS)

Perbandingan hasil *benchmark* (HTTPS) antara *single-node* dan *distributed* (3-node) dengan 100 pengguna konkuren:

1. Latency: Biaya konsensus Raft sangat signifikan.
    - Latensi rata-rata meningkat dari ~893 ms menjadi ~3241 ms (peningkatan ~3.6x).
    - Latensi median (p50) meningkat dari 800 ms menjadi 3200 ms (peningkatan 4.0x).
    - Latensi p95 meningkat dari 1900 ms menjadi 4700 ms (peningkatan ~2.5x). Peningkatan substansial ini adalah harga yang harus dibayar untuk proses replikasi log -> tunggu mayoritas -> commit -> apply yang menjamin konsistensi dan *fault tolerance* dalam Raft.
2. Throughput: Kemampuan sistem memproses *request* per detik menurun drastis.
    - *Throughput* rata-rata turun dari 57.4 RPS menjadi 23.0 RPS (penurunan ~60%). Penurunan signifikan ini menegaskan bahwa *Leader* menjadi *bottleneck* karena harus mengelola komunikasi

8. Kesimpulan

Proyek ini berhasil mengimplementasikan sistem sinkronisasi terdistribusi yang terdiri dari Distributed Lock Manager (Raft), Distributed Queue System (Consistent Hashing + Redis), dan Distributed Cache Coherence (MESI). Fungsionalitas inti dari setiap komponen telah berhasil diverifikasi secara fungsional melalui Unit Testing (menguji logika internal) dan Integration Testing(menguji komunikasi antar komponen melalui HTTPS).

Fitur keamanan dasar berupa enkripsi komunikasi antar node (HTTPS/TLS) dan audit logging yang ditingkatkan juga berhasil diimplementasikan sebagai nilai tambah dan divalidasi melalui integration test.

Analisis performa pada Lock Manager (Raft) dengan HTTPS mengonfirmasi trade-off yang signifikan antara performa dan konsistensi terdistribusi:

- Konfigurasi terdistribusi (3-node) menunjukkan penurunan throughput rata-rata sebesar ~60% (dari 57.4 RPS pada single-node menjadi 23.0 RPS pada 3-node).
- Terjadi peningkatan latensi yang sangat signifikan pada konfigurasi terdistribusi, dengan latensi median naik 4.0x (dari 800 ms menjadi 3200 ms) dan latensi rata-rata naik ~3.6x (dari ~893 ms menjadi ~3241 ms) dibandingkan single-node. Peningkatan latensi p95 juga signifikan (~2.5x, dari 1900 ms menjadi 4700 ms).
- Penurunan performa ini disebabkan oleh overhead yang inheren pada algoritma konsensus Raft, yang memerlukan komunikasi jaringan (replikasi log) dan menunggu konfirmasi dari mayoritas node sebelum sebuah operasi dapat di-commit.
- Meskipun demikian, kedua konfigurasi (single dan 3-node) terbukti stabil dan handal di bawah beban 100 pengguna konkuren, menyelesaikan pengujian dengan 0% failure rate.

Sistem terdistribusi Raft memang tidak membuat aplikasi ini lebih cepat untuk beban kerja write-heavy seperti lock manager. Namun, tujuannya bukan kecepatan, melainkan Fault Tolerance. Sistem 3-node dapat bertahan dari kegagalan 1 node dan melanjutkan operasi secara otomatis, sebuah kemampuan vital yang tidak dimiliki single-node. Laporan ini membuktikan keberhasilan implementasi state machine lock yang fault-tolerant, kini juga diamankan dengan enkripsi dasar, dengan overhead performa yang terukur dan stabil.