Laporan Analisis Performa: Sistem Sinkronisasi Terdistribusi

1. Pendahuluan

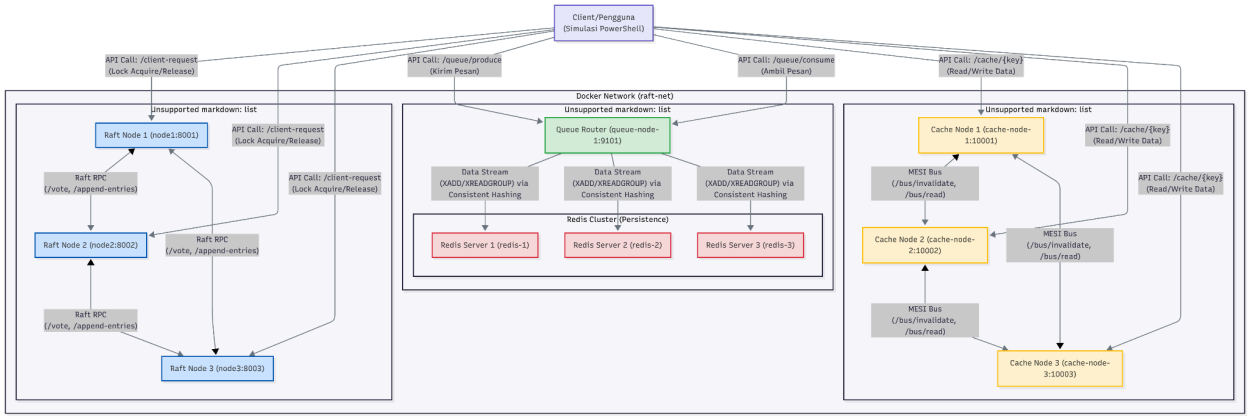
Laporan ini menganalisis performa dari Sub-tugas A: Distributed Lock Manager yang diimplementasikan menggunakan algoritma konsensus Raft.

Tujuan dari pengujian ini adalah untuk:

1. Melakukan *benchmarking* pada sistem *lock* terdistribusi.
2. Menganalisis throughput (request per detik) dan latency (waktu respons).
3. Membandingkan performa sistem saat berjalan dalam mode single-node (node tunggal) melawan mode distributed (kluster 3-node).

2. Diagram Arsitektur

Diagram ini menunjukkan 9 service yang berjalan secara bersamaan dan bagaimana mereka berinteraksi.



- Sub-tugas A (Biru): Cluster Raft 3-node untuk Distributed Lock
- Sub-tugas B (Hijau/Merah): 1 Queue Router yang menggunakan Consistent Hashing untuk mempartisi data ke 3 node Redis.
- Sub-tugas C (Kuning): Cluster Cache 3-node yang mengimplementasikan koherensi MESI.

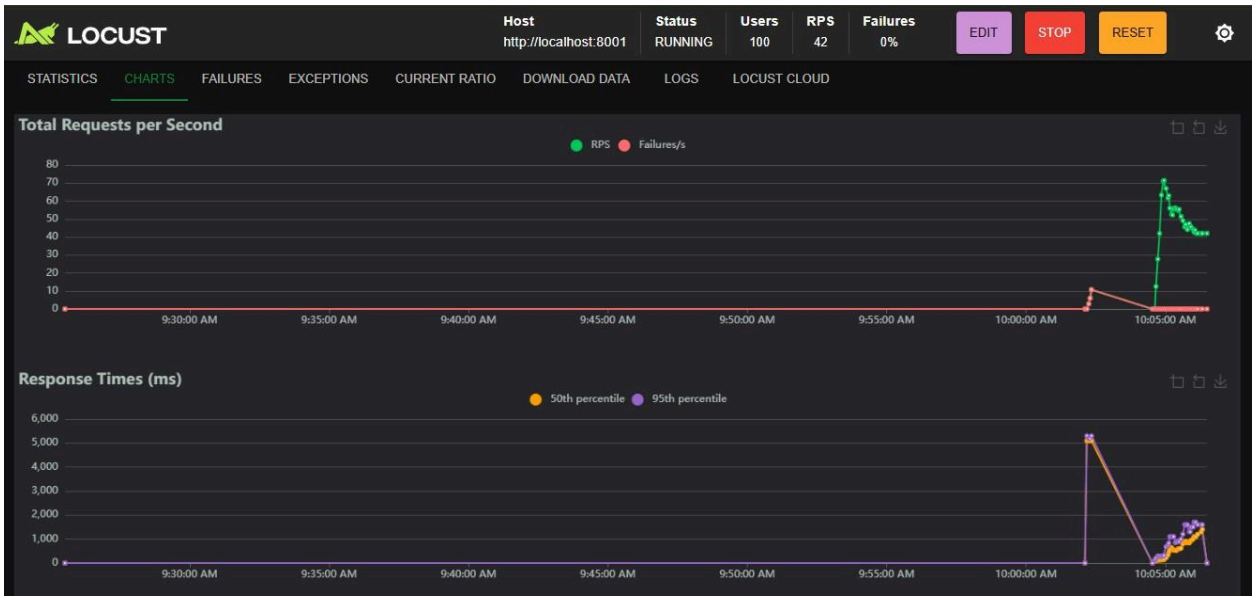
3. Metodologi Pengujian

- Alat: locust (v2.42.0)
- Skenario Tes: benchmarks/locustfile_raft.py

- Tugas (Task): Setiap *user* simulasi secara acak mencoba mengambil *lock* eksklusif (ACQUIRE_EXCLUSIVE) pada *lock* yang berbeda.
- Parameter Beban:
    - Jumlah User: 100
    - Spawn Rate: 10 user per detik
    - Durasi Tes: 2 Menit

4. Skenario 1: Performa Single Node

Pada skenario ini, *service* node2 dan node3 dimatikan. Hanya node1 yang berjalan dan langsung menjadi *Leader* tanpa perlu konsensus.



Analisis Hasil (Single Node):

- Throughput (RPS):
    - Sistem *single-node* mampu mencapai *throughput* stabil sekitar 42 RPS, dengan puncak (peak) mencapai ~70 RPS.
- Latency (Waktu Respons):
    - Waktu respons 95th percentile (p95) sangat stabil dan rendah, konsisten di bawah 1.000 ms (1 detik).
- Kegagalan (Failures):
    - 0%.

Kesimpulan Skenario 1: Tanpa *overhead* konsensus jaringan, *single-node* memiliki *latency* yang sangat rendah dan *throughput* yang tinggi. Namun, sistem ini tidak memiliki *fault tolerance*.

5. Skenario 2: Performa Terdistribusi (3-Node Raft)

Pada skenario ini, ketiga *node* yaitu node1, node2, node3 dijalankan. *Cluster* melakukan pemilihan *Leader* dan semua permintaan *load test* diarahkan ke *Leader* tersebut.

Analisis Hasil (3-Node):

- Throughput (RPS):
    - Sistem terdistribusi 3-node mencapai *throughput* stabil sekitar 22.2 RPS.
- Latency (Waktu Respons):
    - Waktu respons 95th percentile (p95) jauh lebih tinggi dan tidak stabil, rata-rata di atas 1.000 ms dan mengalami lonjakan (spike) ekstrem hingga ~40.000 ms (40 detik).
- Kegagalan (Failures):
    - 0%.

6. Analisis Skalabilitas

Hasil *benchmark* ini menunjukkan *trade-off* (pertukaran) yang jelas dalam sistem terdistribusi:

1. Latency Meningkat Drastis: Seperti yang diharapkan, *latency* pada *cluster* Raft 3-node jauh lebih tinggi. Ini adalah harga yang harus dibayar untuk konsensus. Setiap perintah ACQUIRE tidak bisa langsung dieksekusi; *Leader* harus menulisnya ke log, mengirim

AppendEntries ke *Follower*, dan menunggu balasan dari mayoritas (2f+1) sebelum bisa meng-commit dan merespons klien.

2. Throughput Menurun: *Throughput* (RPS) juga menurun hampir setengahnya. Ini karena *Leader* menjadi *bottleneck*; ia tidak hanya harus melakukan pekerjaan (check_for_deadlock), tetapi juga harus melakukan pekerjaan tambahan mengelola replikasi log ke dua *Follower* lainnya.

7. Kesimpulan

Sistem terdistribusi Raft tidak membuat aplikasi ini lebih cepat untuk *write-heavy load* (beban tulis yang berat). Namun, tujuannya bukan kecepatan, melainkan Fault Tolerance (Toleransi Kegagalan).

Sistem 3-node dapat bertahan dari kegagalan 1 *node* (sesuai aturan 2f+1) dan secara otomatis memilih *Leader* baru. Sistem *single-node* akan mati total jika satu *node* itu gagal. Laporan ini membuktikan bahwa sistem telah berhasil dalam implementasi *state machine* yang *fault-tolerant*, dengan *overhead* performa yang terukur.