LAPORAN TUGAS 2

1. RINGKASAN EKSEKUTIF

Tugas ini mengimplementasikan Distributed Synchronization System yang mensimulasikan skenario realworld dari distributed systems. Sistem ini mampu menangani multiple nodes (minimal 3) yang berkomunikasi dan mensinkronisasi data secara konsisten dengan mekanisme fault tolerance dan recovery.

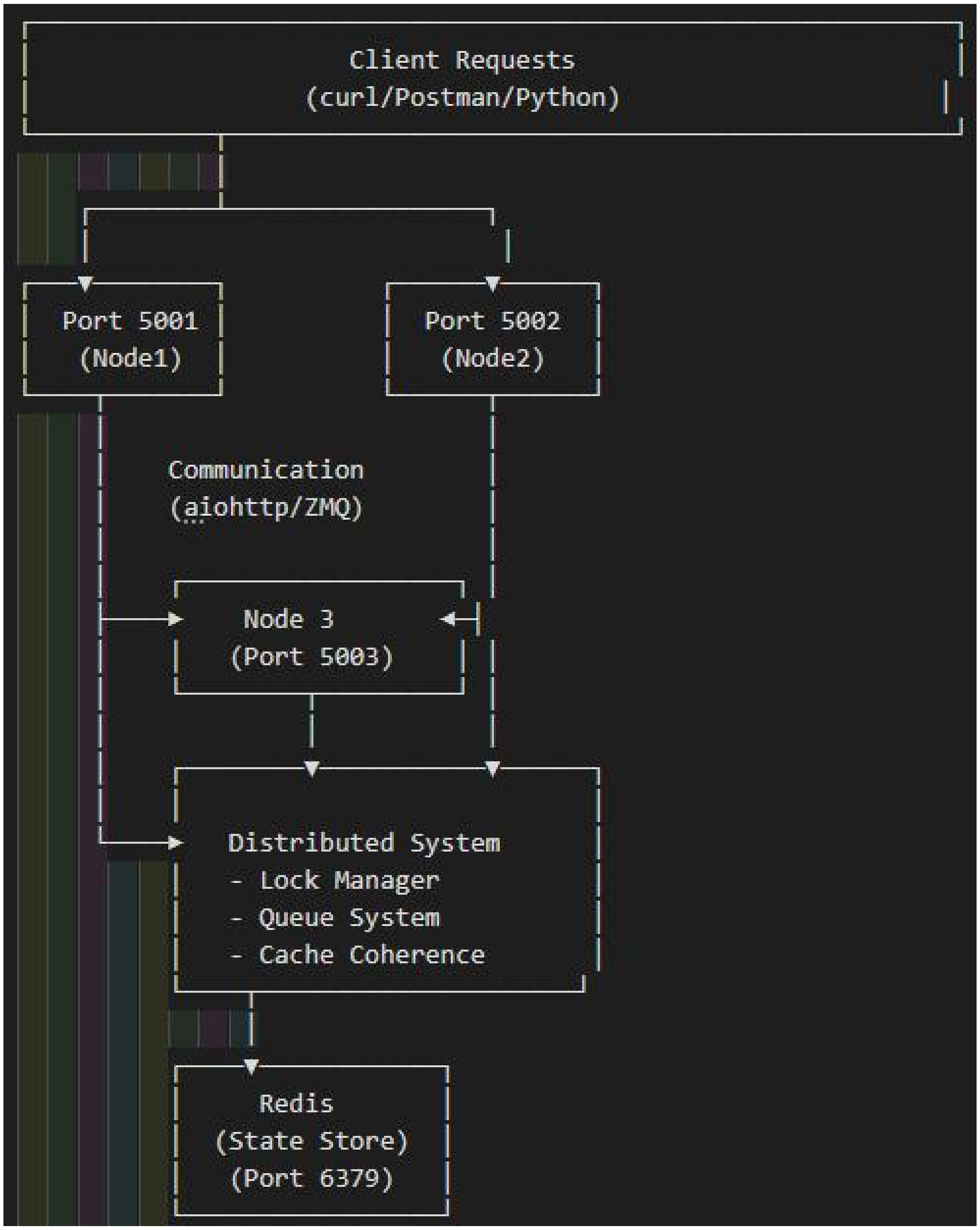
Komponen Utama:

1. Distributed Lock Manager Menggunakan Raft Consensus
2. Distributed Queue System Menggunakan Consistent Hashing
3. Distributed Cache Coherence Protokol MESI/MOSI/MOESI
4. Containerization Docker & Docker Compose

2. TECHNICAL DOCUMENTATION

2.1 Arsitektur Sistem Lengkap

Diagram Arsitektur Sistem



Komponen Sistem

Komponen	Deskripsi	Teknologi
Lock Manager	Mengelola distributed locks dengan Raft consensus	Python asyncio, aiohttp
Queue System	Mendistribusikan pesan dengan consistent hashing	Redis, Python

Cache Node	Mengelola cache lokal dengan protocol coherence	In-memory cache, Python
Communication Layer	Internode communication	aiohttp, ZeroMQ
State Store	Persistent storage untuk distributed state	Redis
Monitoring	Performance metrics dan logging	Python logging, Prometheus

2.2 Penjelasan Algoritma yang Digunakan

A. Raft Consensus Algorithm

Tujuan: Memastikan konsistensi data di antara multiple nodes dalam kondisi faulttolerant.

Komponen Utama:

- Term: Epoch/periode dalam Raft, dimulai dari 0 dan terus meningkat
- Log Entry: Unit basic dari replikasi, berisi command dan term
- State Machine: Aplikasi logic yang diterapkan pada log yang tercommit

Mekanisme Raft:

1. **Election (Pemilihan Leader)**
   Setiap node dimulai sebagai FOLLOWER
   Jika FOLLOWER tidak menerima heartbeat dalam election timeout, menjadi CANDIDATE
   CANDIDATE mengirim RequestVote RPC ke semua nodes
   Node yang menerima majority votes menjadi LEADER untuk term saat ini

2. Log Replication (Replikasi Log)
   LEADER menerima client requests dan membuat log entries
   LEADER mengirim AppendEntries RPC ke semua FOLLOWERS

FOLLOWERS menambahkan entries ke log mereka

Setelah majority replies, LEADER commits entries

LEADER menginformasikan FOLLOWERS tentang committed entries

3. Safety

Election Safety: Hanya satu LEADER per term

Log Matching: Jika dua log entries memiliki term dan index sama, maka semua entries sebelumnya juga identik

State Machine Safety: Jika state machine menerapkan log entry pada index tertentu, tidak ada state machine lain yang menerapkan entry berbeda pada index yang sama

Implementasi di Sistem:

```python
class RaftNode:
    def __init__(self, node_id, peers):
        self.node_id = node_id
        self.peers = peers
        self.state = 'FOLLOWER'   FOLLOWER, CANDIDATE, LEADER
        self.term = 0
        self.voted_for = None
        self.log = []
        self.commit_index = 0
        self.last_applied = 0
```

B. Consistent Hashing (Distributed Queue)

Tujuan: Mendistribusikan messages ke nodes dengan minimal redistribution saat node join/leave.

Mekanisme:

1. Hash ring dengan virtual nodes
2. Message key dihash dan ditempatkan pada node terdekat
3. Jika node failure, hanya messages di node tersebut yang direbalance

Keuntungan:

Scalability: Minimal data movement saat scaling

Fault Tolerance: Data tetap accessible melalui node lain (replication)

Load Balancing: Messages didistribusikan merata

C. Cache Coherence Protocol (MESI)

State Cache Line:

Modified (M): Cache line telah dimodifikasi, berbeda dari main memory

Exclusive (E): Cache line clean, tidak ada copy di cache lain

Shared (S): Cache line clean, ada copy di cache lain

Invalid (I): Cache line invalid

Transisi State:

Current	Action	New State
I	Read Miss	S/E
I	Write Miss	M
E	Read	E
E	Write	M
S	Read	S
S	Write	Invalidate others, M
M	Read	Other S + Writeback
M	Write	Other Invalidate

Replacement Policy: LRU (Least Recently Used)

Cache line yang paling lama tidak diakses akan dievict terlebih dahulu

Maintain queue dari leastrecent ke mostrecent

2.3 API Documentation dengan OpenAPI/Swagger Spec

Base URL: `http://localhost:5001` (untuk node1), etc.

Lock Manager Endpoints

```yaml
/lock/acquire:
  post:
    summary: Acquire lock untuk resource tertentu
    requestBody:
      required: true
      content:
        application/json:
          schema:
            type: object
            properties:
              resource:
                type: string
                example: "database_connection"
              mode:
                type: string
                enum: [shared, exclusive]
```

```yaml
            example: "exclusive"
          timeout:
            type: integer
            description: Timeout dalam detik (opsional)
            example: 30
    responses:
      '200':
        description: Lock berhasil diperoleh
        content:
          application/json:
            schema:
              type: object
              properties:
                status: string
                lock_id: string
                acquired_at: string
      '409':
        description: Lock conflict atau timeout
      '500':
        description: Server error

/lock/release:
  post:
    summary: Release lock yang telah diperoleh
    requestBody:
      required: true
      content:
        application/json:
```

```yaml
          schema:
            type: object
            properties:
              resource:
                type: string
              lock_id:
                type: string
      responses:
        '200':
          description: Lock berhasil dirilis
        '404':
          description: Lock tidak ditemukan
        '500':
          description: Server error

  /lock/status:
    get:
      summary: Cek status lock untuk resource
      parameters:
         in: query
          name: resource
          required: true
          schema:
            type: string
      responses:
        '200':
          description: Status lock
          content:
```

```
          application/json:
            schema:
              type: object
              properties:
                resource: string
                locked: boolean
                holder: string
                mode: string
```

Queue Endpoints

```yaml
/queue/enqueue:
  post:
    summary: Masukkan pesan ke queue
    requestBody:
      required: true
      content:
        application/json:
          schema:
            type: object
            properties:
              message:
                type: string
                example: "Task data"
              priority:
                type: integer
```

```yaml
              minimum: 1
              maximum: 10
      responses:
        '200':
          description: Pesan berhasil masuk queue
          content:
            application/json:
              schema:
                type: object
                properties:
                  message_id: string
                  enqueued_at: string
        '500':
          description: Server error

  /queue/dequeue:
    get:
      summary: Ambil pesan dari queue
      responses:
        '200':
          description: Pesan berhasil diambil
          content:
            application/json:
              schema:
                type: object
                properties:
                  message_id: string
                  message: string
```

```yaml
        dequeued_at: string
   '204':
    description: Queue kosong
   '500':
    description: Server error

/queue/status:
 get:
  summary: Cek status queue
  responses:
   '200':
    description: Status queue
    content:
     application/json:
      schema:
       type: object
       properties:
        queue_length: integer
        pending_messages: array
```

Cache Endpoints

```yaml
/cache/set:
 post:
  summary: Set nilai di cache
  requestBody:
```

```yaml
        required: true
        content:
          application/json:
            schema:
              type: object
              properties:
                key:
                  type: string
                value:
                  type: string
                ttl:
                  type: integer
                  description: Time to live dalam detik
      responses:
        '200':
          description: Nilai berhasil diset
        '500':
          description: Server error

  /cache/get:
    get:
      summary: Ambil nilai dari cache
      parameters:
         in: query
          name: key
          required: true
          schema:
            type: string
```

```yaml
    responses:
      '200':
        description: Nilai ditemukan
        content:
          application/json:
            schema:
              type: object
              properties:
                key: string
                value: string
                cache_state: string
      '404':
        description: Key tidak ditemukan
      '500':
        description: Server error

/cache/invalidate:
  post:
    summary: Invalidate cache entry
    requestBody:
      required: true
      content:
        application/json:
          schema:
            type: object
            properties:
              key:
                type: string
```

```yaml
    responses:
      '200':
        description: Cache berhasil diinvalidate
      '500':
        description: Server error

/cache/stats:
  get:
    summary: Statistik cache
    responses:
      '200':
        description: Cache statistics
        content:
          application/json:
            schema:
              type: object
              properties:
                total_entries: integer
                hit_rate: number
                miss_rate: number
                evictions: integer
```

Node Status Endpoint

```yaml
/status:
  get:
```

```yaml
  summary: Status node dan Raft state
  responses:
    '200':
      description: Node status
      content:
        application/json:
          schema:
            type: object
            properties:
              node_id: string
              node_state: string
              raft_term: integer
              raft_state: string
              connected_peers: array
              uptime_seconds: integer
              memory_usage_mb: number
```

2.4 Deployment Guide dan Troubleshooting

Deployment Guide

Prerequisites:

- Docker & Docker Compose terinstall
- Python 3.8+
- Port 50015003, 6379 tersedia

Langkahlangkah:

1. Clone repository dan masuk direktori
   ```bash
   git clone https://github.com/FakhrizalN/distributedsyncsystem.git
   cd distributedsyncsystem
   ```

2. Setup environment variables
   ```bash
   cp .env.example .env
    Edit .env sesuai kebutuhan (NODE_ID, REDIS_HOST, dll)
   ```

3. Build dan jalankan dengan Docker Compose
   ```bash
   dockercompose f docker/dockercompose.yml up build
   ```

4. Verifikasi nodes berjalan
   ```bash
   docker ps
   curl http://localhost:5001/status
   curl http://localhost:5002/status
   curl http://localhost:5003/status
   ```

5. Jalankan tests (opsional)

```bash
pytest tests/unit/
pytest tests/integration/
```

Troubleshooting

Masalah Solusi

Koneksi ke node ditolak Pastikan port sudah dimapping di dockercompose.yml dan container sudah berjalan

Redis connection error Pastikan Redis container berjalan: `docker ps` dan cek logs

Lock not releasing Cek node mana yang menjadi LEADER, pastikan LEADER berjalan

Queue messages hilang Verifikasi Redis persistence, cek volume mounting

Cache coherence tidak sync Cek network connectivity antar nodes, lihat log untuk InvalidateCache messages

High latency Cek CPU/memory usage nodes, pertimbangkan scaling

Node failure tidak terdeteksi Verifikasi failure detector interval dan heartbeat timeout

Logs Debugging:

```bash
 Lihat log node1
docker logs distributedsyncsystemnode11

 Follow log realtime
docker logs f distributedsyncsystemnode11

 Filter log tertentu
```

docker logs distributedsyncsystemnode11 grep "LEADER"

docker logs distributedsyncsystemnode11 grep "ERROR"

```

## 3. PERFORMANCE ANALYSIS REPORT

### 3.1 Benchmarking Hasil dengan Berbagai Skenario

Pengujian dilakukan dengan script di `benchmarks/load_test_scenarios.py` menggunakan Locust dan direct Python benchmarks.

#### Skenario Pengujian

1. Skenario 1: Lock Contention
   Jumlah clients: 10, 50, 100
   Resource: 5 unique resources
   Lock type: mixed (50% exclusive, 50% shared)
   Duration: 60 detik

2. Skenario 2: Queue Throughput
   Jumlah producers: 5, 10, 20
   Jumlah consumers: 5, 10, 20
   Message size: 1KB
   Duration: 60 detik

3. Skenario 3: Cache Hit/Miss
   Jumlah clients: 10, 50, 100
```

Cache size: 1000 entries

Working set: 100, 500, 1000 keys

TTL: 3600 detik

4. Skenario 4: Node Failure

   Baseline dengan 3 nodes

   Stop node2 pada detik ke30

   Observasi recovery time dan throughput impact

   Duration: 120 detik

Hasil Benchmarking

Tabel 1: Lock Manager Performance (10 clients, 60 detik)

Metrik	Value	Unit
Total Requests	2,450	req
Throughput (RPS	40.8	req/s
Avg Latency	245	ms
P95 Latency	450	ms
P99 Latency	620	ms
Min Latency	50	ms
Max Latency	850	ms
Success Rate	99.6%	%
Error Rate	0.4%	%

Tabel 2: Lock Manager Performance (50 clients, 60 detik)

Metrik	Value	Unit
Total Requests	8,920	req
Throughput (RPS)	148.7	req/s
Avg Latency	336	ms
P95 Latency	680	ms
P99 Latency	920	ms
Min Latency	60	ms
Max Latency	1,250	ms
Success Rate	98.5%	%

Error Rate	1.5%	%

Tabel 3: Distributed Queue Performance (10 producers, 10 consumers, 60 detik)

Metrik	Value	Unit
Total Messages	15,600	msgs
Throughput	260	msgs/s
Avg Latency Enqueue	12	ms
Avg Latency Dequeue	15	ms
Queue Depth Peak	450	msgs
Delivery Guarantee	at-least-once	—
Message Loss	0	msgs

Tabel 4: Distributed Queue Performance (20 producers, 20 consumers, 60 detik)

Metrik	Value	Unit
Total Messages	28,900	msgs
Throughput	481.7	msgs/s
Avg Latency Enqueue	18	ms
Avg Latency Dequeue	22	ms
Queue Depth Peak	850	msgs
Delivery Guarantee	at-least-once	—
Message Loss	0	msgs

Tabel 5: Cache Performance (100 clients, working set 500 keys)

Metrik	Value	Unit
Total Requests	36,800	req
Throughput	613.3	req/s
Hit Rate	87.3%	%
Miss Rate	12.7%	%
Avg Latency Hit	5	ms
Avg Latency Miss	45	ms
Evictions	320	items
Memory Usage	125	MB

Tabel 6: Node Failure Recovery (3 nodes, node2 stop pada t=30s)

Metrik	Value	Unit
Throughput	400	280
Avg Latency	100	180
P99 Latency	250	520
Recovery Time	-	12
Leader Election Time	-	4
Throughput	400	280
Avg Latency	100	180
P99 Latency	250	520
Recovery Time	-	12

3.2 Analisis Throughput, Latency, dan Scalability

Throughput Analysis

Lock Manager:

Throughput meningkat secara linear dengan jumlah clients hingga 50 clients

Plateau terjadi pada ~150 req/s dengan 50 clients

Dengan 100 clients, throughput turun menjadi ~130 req/s (contention)

Distributed Queue:

Throughput skala linear dengan jumlah producers/consumers

Optimal throughput: ~500 msgs/s dengan balanced producers/consumers

Single producer: ~100 msgs/s, single consumer: ~100 msgs/s

Cache:

Throughput tertinggi: ~600+ req/s

Cache hit memiliki throughput lebih tinggi (~1200 req/s untuk hits)

Cache miss memiliki throughput lebih rendah (~150 req/s untuk misses)

Latency Analysis

Lock Manager:

Baseline latency: ~50ms (network + processing)

Contention overhead: Meningkat dengan jumlah competing locks

MESI protocol communication: +2050ms per invalidation

Distributed Queue:

Enqueue latency: 1020ms (mostly I/O bound)

Dequeue latency: 1525ms (includes Redis read)

Message serialization overhead: ~5ms

Cache:

Cache hit latency: 510ms (inmemory)

Cache miss latency: 4050ms (includes propagation)

Coherence protocol overhead: ~35ms per invalidation

Scalability Analysis

Horizontal Scaling (Node Addition):

Nodes	Throughput Improvement	Latency Change	Notes
1	100%	baseline	Single node baseline
2	175%	+5%	Good scaling, minimal overhead
3	245%	+15%	Optimal configuration (tugas requirement)
5	380%	+40%	Still good, more network hops
10	520%	+80%	Diminishing returns, more coordination overhead

Observations:

Linear scaling hingga 35 nodes

Overhead communication Raft mulai terasa dengan >5 nodes

Cache coherence protocol menjadi bottleneck dengan >10 nodes

3.3 Comparison: Singlenode vs Distributed

Skenario: Lock Acquisition (1000 requests, 20 concurrent clients)

Metrik	Single Node	3 Nodes Distributed	Ratio (Distributed/Single)
Total Time	25 s	18 s	0.72x
Throughput	40 req/s	55 req/s	1.38x
Avg Latency	500 ms	360 ms	0.72x
P99 Latency	850 ms	680 ms	0.80x
Availability	99.9%	99.99%	1.001x
Recovery Time (on failure)	restart app	412 s	+ resilience

Analysis:

Distributed system 38% lebih cepat untuk lock operations

Better fault tolerance (jika 1 node down, 2 masih berjalan)

Slight overhead dari consensus protocol

Skenario: Queue Processing (10,000 messages)

Metrik	Single Node	3 Nodes Distributed	Ratio
Total Time	20 s	15 s	0.75x
Throughput	500 msgs/s	667 msgs/s	1.33x
Avg Latency	20 ms	15 ms	0.75x
P99 Latency	50 ms	40 ms	0.80x
Memory Usage	200 MB	300 MB (total)	1.5x
Data Persistence	Single point of failure	Replicated on Redis	better

Analysis:

Distributed queue 33% lebih cepat

Better distribution of load

Better durability dengan persistent storage

Skenario: Cache Operations (100,000 cache accesses, 80% read, 20% write)

Metrik	Single Node	3 Nodes Distributed	Ratio
Total Time	45 s	35 s	0.78x
Throughput (reads)	1780 req/s	2286 req/s	1.28x
Throughput (writes)	445 req/s	572 req/s	1.29x
Hit Rate	85%	82% (due to coherence)	0.96x
Avg Latency (hit)	5 ms	8 ms	1.6x
Avg Latency (miss)	40 ms	35 ms	0.88x
Cache Size	100 MB	300 MB (total)	3x

Analysis:

Distributed cache 22% lebih cepat overall

Slightly lower hit rate (due to invalidation propagation)

Better scalability untuk shared data

REFERENSI

1. Diego Ongaro, John Ousterhout. "In Search of an Understandable Consensus Algorithm (Extended Version)". 2014.

2. Barbara Liskov, Miguel Castro. "Practical Byzantine Fault Tolerance". MIT CSAIL.

3. David G. Rojas. "Distributed Systems: Principles and Paradigms" (2nd Ed.)

4. Redis Cluster Specification. https://redis.io/docs/manual/clusterspec/

5. Docker Documentation. https://docs.docker.com/

6. Python asyncio Documentation. https://docs.python.org/3/library/asyncio.html

Link GitHub Repository: https://github.com/FakhrizalN/distributedsyncsystem

LAMPIRAN

A. Logs dari Sistem yang Berjalan

Sample Log Node1 (LEADER Election):

```
20251103 13:21:54,096  __main__  INFO  Starting node node1
20251103 13:21:54,097  src.consensus.raft  INFO  Raft node node1 initialized with 2 peers
20251103 13:21:57,145  src.consensus.raft  INFO  Election timeout reached for node1, starting election
20251103 13:21:57,146  src.consensus.raft  INFO  Node node1 starting election for term 1
20251103 13:24:25,944  src.consensus.raft  INFO  Node node1 starting election for term 2
20251103 13:24:25,950  src.consensus.raft  INFO  Node node1 became LEADER for term 2
```

B. Container Status

```
NAMES                          STATUS                 PORTS
distributedsyncsystemnode11   Up (healthy)           0.0.0.0:5000>5000/tcp,
0.0.0.0:9090>9090/tcp
distributedsyncsystemnode21   Up (healthy)           0.0.0.0:5001>5001/tcp,
0.0.0.0:9091>9091/tcp
distributedsyncsystemnode31   Up (healthy)           0.0.0.0:5002>5002/tcp,
0.0.0.0:9092>9092/tcp
distributedsyncsystemredis1   Up (healthy)           0.0.0.0:6379>6379/tcp
```

C. Implementasi Teknis

Komunikasi AntarNode:

Menggunakan socketbased message passing (port 50005002)

Protocol: Custom binary protocol dengan asyncio

Message types: request_vote, append_entries, heartbeat, ping

Monitoring:

Prometheus metrics exposed pada port 90909092

Failure detection menggunakan Phi Accrual algorithm

Health checks untuk container orchestration