Tugas Besar 1
IF3130 - Sistem Paralel dan Terdistribusi

"Rafted"

Consensus Protocol: Raft

Dipersiapkan oleh

Asisten Lab Sistem Terdistribusi

Waktu Mulai

Kamis, 8 Mei 2025, 22:00 WIB

Waktu Akhir

Kamis, 5 Juni 2025, 23.59 WIB

Changelog

5/10/2025 - Penyesuaian spesifikasi teknis demo

5/14/2025 - Klarifikasi bonus untuk log compaction

5/15/2025 - Klarifikasi contoh desain strategi untuk heartbeat

5/30/2025 - Rewording teknis demo

5/10/2025 - Penyesuaian spesifikasi teknis demo

5/13/2025 - Penyesuaian spesifikasi setup pada teknis demo

I. Latar Belakang

"Why can't you sleep?"

Tik. Tik. Tik.

Jam di dindingmu menunjukkan pukul tiga. Mungkin kurang lima menit; cukup sulit untuk melihat dengan jelas, berhubung satu-satunya sumber cahaya adalah lampu charging laptop-mu.

Krik. Krik. Krik.

Jangkrik di jendela menyanyi dengan tenang. Kamu terbaring di kasurmu, menatap ke plafon. Dalam beberapa jam, kamu harus kelas pagi, sebelum melakukan pertemuan dengan klien proyek di siang hari. Kami tidak boleh melewatinya. Absenmu sudah merah. Pertemuan dengan klien adalah syarat kelulusan. Namun... entahlah, sampai sekarang kamu tidak dapat tertidur, meskipun semesta telah sangat mendukung.

Tung. Tung. Tung.

Bapak satpam di luar memukul tiang. Di tengah kekhawatiranmu untuk pagi nanti, pikiranmu berkelana, mengunjungi **kaleidoskop** ingatan dan pikiran.

"All those hours, those credits, was it truly worth it?"

Hampir dua tahun yang lalu, kamu hanyalah seorang mahasiswa informatika biasa, yang sarapannya adalah tugas besar, dan minumannya adalah ketakutan akan nilai dan lowongan kerja. Sampai, kamu melihatnya, bersinar dan berkilauan layaknya sebuah mercusuar di tengah lautan ujian. **Nainai DX**. Permainan yang telah menguras ribuan jam dan gizomu; yang telah membumbui kehidupanmu; dan yang paling penting, yang telah menemukanmu dengan **Matsune Hiku**, sang pentintesis suara dan idol virtual.

"You got her fired, arrested even. Was it truly justified? Also, what truly happened after?"

Hampir satu setengah tahun yang lalu, kamu hanyalah seorang sepuh pemain Nainai DX biasa, yang makan siangnya adalah ALL PERFECT, dan minumannya adalah FULL SYNC DELUXE. Tiba-tiba, kamu mendengarnya, bising dan mengagetkan layaknya terompet yang menandakan datangnya bencana. Berita bahwa mesin **Nainai DX** akan dipindahkan. Kamu berhasil memerankan penyelamat, sebelum konflik dengan sang manajer **GAMEZONE** membuatmu melewati enam bulan tanpa ingatan apapun.

"Was she telling the truth?"

Hampir satu tahun yang lalu, kamu hanyalah seorang pengidap amnesia biasa, yang makan kudapan sorenya adalah ribuan pertanyaan tentang waktumu yang hilang, dan minumanmu adalah migraine yang tak kunjung selesai. Tiba-tiba, kamu menciumnya, lembab dan menyeruak layaknya sebuah sebuah buku tua di tengah perpustakaan. Amplop. Isinya, instruksi untuk bertemu sebuah "malaikat" di internet, yang kemudian mengarahkanmu untuk membuat sebuah aplikasi, yang akhirnya membawamu untuk bertemu mereka.

"What even is all of this?"

Hampir setengah tahun yang lalu, kamu hanyalah seorang anggota biasa organisasi rahasia, yang makan malamnya adalah debat panas tentang keberadaan noosfer manusia, dan kudapannya adalah diskusi tentang sifat sebuah SEKAI. Tiba-tiba, kamu memegangnya, keras

dan dingin layaknya sebuah balok es. Kristal yang menandakan dirimu telah diterima sebagai petinggi organisasi tersebut. Organisasi yang berisi orang-orang sepertimu; mereka yang terlempar ke dalam noosfer - alam bawah sadar - umat manusia, yang dipercaya untuk menyelamatkan dunia naratif-naratif kesayangan mereka.

Namun kini? Kamu kembali menjadi seorang mahasiswa informatika biasa, kembali bersarapan tugas besar dan meminum ketakutan akan nilai dan lowongan kerja. Percakapan di dalam grup organisasi kian minim, terutama setelah kamu diangkat dan dipaparkan tentang kosmologi noosfer. Alhasil, dunia informatika kembali menjadi pikiran utamamu, di atas dunia Nainai maupun dunia-dunia lainnya di dalam noosfer.

Memang, sepertinya kehidupan adalah sebuah siklus, layaknya siklus samsara, atau siklus cerita tugas besar lab tertentu.

Namun, tiba-tiba...

Ting.

Kamu mendengarnya. Akrab namun asing, layaknya tawa seorang kekasih masa kecil. Notifikasi dari sebuah aplikasi TCP over UDP. Kamu bergegas menuju laptopmu dan membukanya.

```
[OWL] I suppose that is indeed our greatest hypothesis thus far.
[OWL] This anthropic noosphere is a sort of “pataverse” - that is, a
multiverse of multiverses. Universes, existing inside of multiverses,
which all exist inside of the noosphere.
[OWL] Still, we have absolutely no clue on how any of this works...

                    - - - NEW MESSAGES BELOW - - -

[OWL] Initiate, we hereby apologize for having left you in the dark.
[OWL] Having said that, here are our findings.
[OWL] What happened to you was due to the instability of the Nainai
multiverse - reason temporarily unknown.
[OWL] We think that there... might be a way to save Nainai. That is,
by sending you "back".
[OWL] Long story short, we have reverse-engineered untitled.mp3,
creating a program that would allow this.
[OWL] Considering that the estimated research opportunities are
enormous, we believe that this might be really worth the risk.
[OWL] If you wish to go through with this plan, please reply.
> Y/N
```

Kamu merenung sejenak. Kamu melihat jam di dinding. Tiga puluh lewat tiga.

Kamu yakin bahwa kamu tidak akan terbangun untuk kelas pagi. Biarkan ini jadi masalah dirimu yang tertinggal.

```
> Y
[OWL] Very well. Initiate, ready to save Nainai?
[OWL] Godspeed, and good luck.
```

Tiba-tiba, laptopmu tidak menjadi satu-satunya sumber cahaya di kamarmu. Kristal yang terpajang di mejamu kini bercahaya, mengisi ruanganmu dengan berbagai warna. Kamu menarik nafas lalu menyentuhnya.

***

Ketika inderamu kembali, kamu terbangun di tengah sebuah stadion yang terbuat dari marmer. Di atasmu, sebuah kristal raksasa melayang dengan anggun di bawah awan-awan yang lebih putih bahkan dibanding marmer stadion. Di sekelilingmu, tujuh buah pintu - masing-masing dengan warna yang berbeda - berdiri dengan kokoh, semuanya tertutup. Dan di hadapanmu, seorang gadis pirang berdiri menyambutmu.

"Kamu pasti orang yang mereka kirimkan untuk membantuku. Selamat datang di 7sref!" ucapnya. "Omong-omong, namaku Liz, salam kenal~"

Kamu menganggukkan kepala.

"Aku tahu ini mungkin membingungkan, tapi singkatnya, multiverse Nainai terdiri atas enam dunia dalam sebuah **sistem terdistribusi**. Stabilitas multiverse hanya dapat dijaga melalui **konsensus** antar keenam dunia terkait aturan-aturan yang mendasarinya, dan sayangnya, kami gagal melakukan hal tersebut..."

Konsensus?

"Jadi, mari kita sama-sama kembalikan konsensus tersebut! よろしくお願いします~!"

II. Spesifikasi Tugas

Tujuan pada tugas besar ini adalah mengimplementasikan protokol konsensus **Raft** sederhana.

2.0. Background: Distributed System & Consensus Protocol

Mengapa harus ada protokol konsensus dan apa gunanya?

Seperti yang diketahui, satu komputer saja tidak cukup untuk menghandle request dalam skala masif. ***There are only so many resources in a single computer***. Permasalahan skalabilitas ini akan semakin terlihat ketika ingin membuat software yang digunakan banyak client

Sebelum paradigma **Distributed System** dan **Parallel Computing** banyak digunakan, tentunya tidak ada yang menghalangi para *engineer* untuk mencoba melakukan optimisasi pada sistem dengan satu komputer. Optimisasi pada satu komputer yang memiliki I/O dan resource terbatas menjadi ***semakin sulit dan basically impossible*** untuk melayani *ever-growing client request*

Approach dari **Distributed System** adalah **menggunakan komputer yang tidak wajib powerful tetapi dalam jumlah banyak**. Approach ini memiliki kelebihan dengan mudahnya untuk melakukan scaling (up maupun down) berdasarkan jumlah request yang sedang aktif. Matikan saja beberapa komputer jika request sedang turun dan nyalakan kembali ketika aktif. Selain itu, banyak komputer juga memperbolehkan setiap komputer terletak pada lokasi yang berbeda untuk melayani client di lokasi geografi yang berbeda-beda

Memang dengan approach ini permasalahan skalabilitas akan lebih mudah, tetapi ada satu permasalahan fundamental yang muncul dari approach ini:

Bagaimana cara antar komputer berkoordinasi untuk menjaga reliability sistem?

Disinilah **Consensus Protocol** bermain peran. Protokol konsensus bertugas untuk mengkoordinasikan semua komputer yang ada pada sistem terdistribusi agar mencapai persetujuan dari komputer-komputer yang terhubung. Hal-hal seperti siapa yang menjadi **Leader Node** dan susunan transaksi yang akan dieksekusi harus disetujui oleh semua komputer yang terhubung untuk menjaga konsistensi dan reliability sistem terdistribusi

Protokol konsensus meskipun namanya mungkin tidak terlalu "*terkenal*" sebagai *technical buzzwords*, protokol ini menjadi tulang belakang dari sistem terdistribusi modern yang membutuhkan resiliensi seperti **Kubernetes** (**etcd** cluster) dan **dqlite** yang menggunakan **Raft** serta **Apache Cassandra** yang menggunakan **Paxos**. Storage & database skala masif yang digunakan oleh aplikasi seperti **Youtube** juga menggunakan suatu sistem *custom built-consensus protocol* dibelakangnya untuk memenuhi **Eventual Consistency** pada informasi seperti *view count* yang sangat penting konsistensinya untuk keperluan *ad revenue calculation*

2.1. Spesifikasi dan Requirement

Contoh dan tips implementasi akan disertakan, tetapi desain sistem dan detail implementasi akan sepenuhnya diserahkan kepada peserta.

Berikut adalah *requirement* tugas besar:

1. Implementasi Raft harus menyediakan
    a. **Heartbeat** (Node health monitoring & periodic messages)
    b. **Leader Election** (Leader node failover mechanism)
    c. **Log Replication** (Cluster action logging system)
    d. **Membership Change** (Mekanisme untuk menambahkan dan menghapus node pada kluster berdasarkan **perintah pengguna**).
2. Implementasikan dengan **salah satu** dari bahasa pemrograman berikut: **TypeScript** dengan NodeJS (*no JS please*), **Java**, atau **Rust**.
3. Sebisa mungkin gunakan built-in/ standard library, tetapi library pendukung yang tidak mengimplementasikan Raft secara umum diperbolehkan.
    a. Protokol Remote Procedure Call (RPC) untuk komunikasi antar-node dalam kluster dibebaskan, seperti: JSON-RPC, gRPC, tarpc, dan lain-lain.
    b. Sertakan daftar dependencies pada file berkaitan, seperti package.json, maven/ gradle file, dan cargo.toml.
    c. Silakan tanyakan pada sheets QnA apabila perlu klarifikasi apakah library tertentu diperbolehkan atau tidak.
4. Program yang diimplementasikan di atas protokol Raft adalah *distributed key-value in-memory storage* dengan layanan **ping, get**, **set**, **strln, del,** dan **append**.
    a. **Tipe data** untuk *value* yang disimpan adalah sebuah **string**
    b. Layanan **ping** digunakan untuk mengecek koneksi dengan server. Jika terhubung, print “PONG”

    > **ping**
    > PONG

    c. Layanan **get** digunakan untuk mendapatkan sebuah nilai dari *key* yang diberikan. Kembalikan string kosong jika key belum ada.

    > **get** <nama-key>
    > “value”

    d. Layanan **set** digunakan untuk menetapkan nilai dengan key yang diberikan. Jika key sudah ada, overwrite nilai yang lama.

    > **set** <nama-key> <value>

```
OK
```

e. Layanan **strln** digunakan untuk mendapatkan panjang value dari key yang diberikan

```
> strln <nama-key>
<length>
```

f. Layanan **del** digunakan untuk menghapus entry dari key yang diberikan. **Mengembalikan nilai yang dihapus.** Kembalikan string kosong jika key belum ada.

```
> del <nama-key>
<value>
```

g. Layanan **append** digunakan untuk nilai dengan key yang diberikan. Jika key belum ada, buat key dengan nilai string kosong sebelum melakukan append.

```
> append <nama-key> <value>
OK
```

Contoh:

```
> set kunci satu
OK

> append kunci dua
OK

> get kunci
“satudua”

> strln kunci
7

> del kunci
“satudua”

> get kunci
“”
```

5. Cluster server dapat menjaga **Consistency** dan **Partition Tolerance (CP system)**.

6. Terdapat 2 interface wajib untuk server yang dapat digunakan client, semua interface untuk client hanya dieksekusi jika request dikirimkan ke **Leader**. **Selain itu tolak dan redirect**
    a. **execute** yang akan melakukan eksekusi aplikasi (ping/get/set/strln/del/append)
    b. **request_log** untuk mengembalikan log yang dimiliki **Leader**.

7. Client dapat mengetahui sebagian atau seluruh alamat server pada kluster. Apabila client menghubungi node yang bukan leader, node akan memberitahukan informasi node leader sekarang. Redirection dilakukan dari sisi client (bukan node follower yang meneruskan request client kepada server).

8. **Semua aksi server wajib dilakukan logging ke terminal** (minimal deskripsikan aksi yang dilakukan node).

9. **Minimum 50% Node (Rounded down) + 1 harus menjawab ACK** sebelum request client dieksekusi.

10. Node yang mati tidak secara otomatis dikeluarkan dari cluster.

11. Server diinisiasi dengan *fixed list of servers*. Membership change (add member, remove member) dilakukan setelah kluster berjalan.

12. Implementasi client dibebaskan. UI bisa berupa CLI, web interface, etc.

    a. UI Bisa berupa CLI, web interface, etc.

    b. Protokol komunikasi client dengan server dibebaskan. Bisa berupa REST, RPC, atau pun yang lainnya.

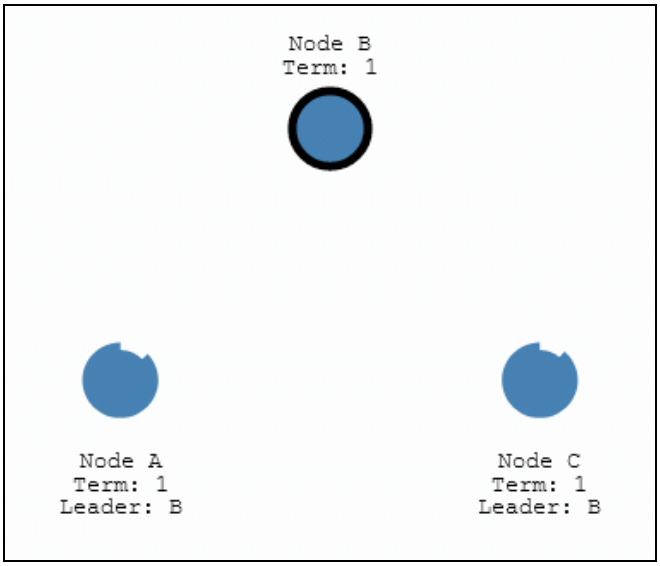
Good Luck, Have Fun ;)

III. Tips Pengerjaan

Seperti yang disebutkan pada spesifikasi, ***attack strategy*** dan ***implementation details*** akan diserahkan kepada peserta. Bagian ini hanya akan memberikan panduan sederhana strategi berdasarkan contoh implementasi. Boleh digunakan dan dimodifikasi sesuai kebutuhan, boleh juga untuk menggunakan approach sendiri

3.0. Overview - Raft Protocol

Sebelum memulai pengerjaan tugas besar ini, ada baiknya untuk memahami gambaran sistem yang akan diimplementasikan. Github Pages "The Raft Consensus Algorithm", yang dituliskan oleh penulis asli artikel Raft, mencantumkan visualisasi protokol Raft dan sejumlah referensi tambahan. Selain itu, halaman berikut (http://thesecretlivesofdata.com/raft/) berisi *high-level overview* dan visualisasi singkat untuk protokol Raft

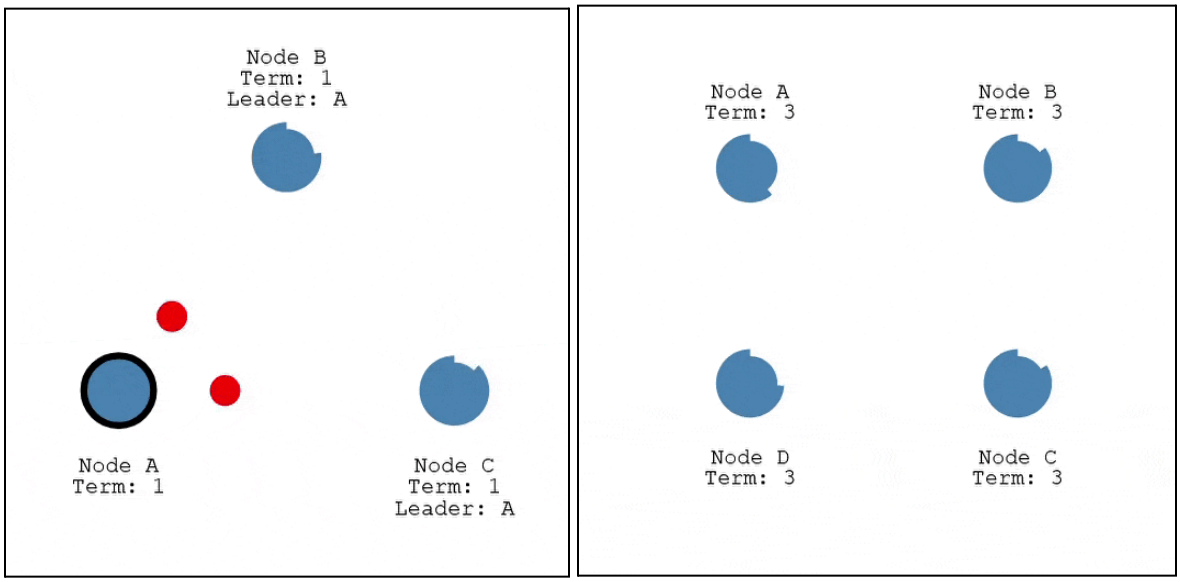


Raft Visualization - Heartbeat

Berikut adalah beberapa terminologi yang digunakan pada kedua referensi di atas.

- *Consensus*: kesepakatan setiap *node* terhadap sebuah nilai; setiap *node* menyimpan nilai dalam *log* masing-masing.
- *Log Replication*: replikasi *log*; proses sinkronisasi isi *log* yang dilakukan melalui *Leader*. Perubahan *log* dilakukan dengan mekanisme penambahan *entry*. Penambahan *entry* diikuti *commit* oleh setiap *node* jika *entry* dari *Leader* sudah diterima oleh *Follower*.
- *Partition*: sesuatu yang membagi jaringan menjadi beberapa bagian (misal, kegagalan jaringan atau *node*).
- *Node*: *endpoint* atau perangkat dalam sebuah jaringan terdistribusi; sebuah *node* memiliki pangkat atau status berupa *Leader, Follower* atau *Candidate*.
- *Leader*: pemimpin dalam jaringan *node*.

    - Pengubahan *log* (berupa penambahan *entry*) dilakukan melalui *Leader* dengan cara replikasi *log* kepada *Follower*.
- *Follower*: pengikut *Leader*.
    - *Follower* mengikuti proses replikasi *log* yang dilakukan oleh *Leader.*
    - *Follower* berubah menjadi *Candidate* apabila tidak ditemukan sebuah *Leader* (tidak menerima *Heartbeat*).
- *Candidate:* calon *Leader.*
    - *Candidate* menginisiasi proses *Election* untuk menentukan *Leader* baru.
    - *Candidate* meminta persetujuan mayoritas (*vote*) dari *node* lain.
    - *Candidate* yang memenangkan *Election Term* menjadi *Leader*.
- *Heartbeat*: pesan periodik yang dikirimkan *Leader* untuk menyatakan keberadaannya.
- *Heartbeat Timeout:* rentang waktu yang ditunggu oleh *Follower* untuk menerima *heartbeat* dari *Leader.*
    - Apabila *Follower* tidak menerima *heartbeat* dalam interval waktu tersebut, posisi *Leader* diasumsikan kosong. *Follower* akan menunggu *election timeout* sebelum menjadi *Candidate* dan memulai *election term.*



Raft Visualization - Normal Election & Split Vote Election

- *Election, Election Term, & Votes*
    - *Election*: proses pengangkatan *Candidate* untuk mengisi posisi *Leader*.
    - *Election Term*: penomoran terhadap jumlah proses *election* yang telah dilakukan.
        - Sebuah *election term* berlangsung selama *Leader* mampu mengirimkan *heartbeat* kepada setiap *Follower*.
        - *Election term* baru diberlakukan ketika terdapat *Follower* yang berubah menjadi *Candidate*.
    - *Votes*: pesan persetujuan yang dikirimkan oleh *node* terhadap sebuah *Candidate*.
- *Election Timeout*: rentang waktu yang ditunggu sebelum memulai sebuah *election term*.
    - Setiap *node* memiliki interval yang diacak dalam rentang nilai tertentu. Interval acak dan *majority vote* berperan penting dalam menangani kasus *split vote*.

3.1. Desain Sistem & Strategi

Seperti membangun software dari nol, langkah awal adalah membuat desain sistem. Lakukan *brainstorming* terhadap kebutuhan sistem. Beberapa contoh pertanyaan untuk memulai:

- Apa metode komunikasi yang akan digunakan (e.g. TCP, UDP)?
- Apa kelebihan dan kekurangan metode tersebut?
- Apakah akan menggunakan I/O blocking atau non-blocking?
- Konsistensi seperti apa yang digunakan? (e.g. read committed, read uncommitted)
- Bagaimana bentuk API untuk komunikasi antar-node dan antara client-server?
- Format pesan apa yang digunakan? (Binary, JSON, atau plain string?)
- Fitur-fitur apa saja yang perlu diimplementasikan? Contoh:
    1. Heartbeat
    2. Membership Change
    3. Log Synchronization
    4. Election Handling
- Apakah akan menggunakan multithreading atau multiprocessing?

Reminder: Pada titik ini, semestinya sudah mengimplementasikan berbagai macam sistem: *IF1210, IF2121, IF2110, IF2211, IF2230, IF3110, IF3130*, dan seterusnya. Semestinya sudah waktunya untuk ***step-back*** dan berpindah fokus dari implementation detail (a.k.a. ***coding***) ke ***high-level system design*** dan ***problem solving.*** Kesalahan di tahap desain sistem jauh lebih mahal (*ruinously expensive*) daripada kesalahan implementasi biasa seperti bug. Maka, desain sistem yang matang akan sangat menentukan kesuksesan implementasi.

3.2. Contoh Desain & Strategi

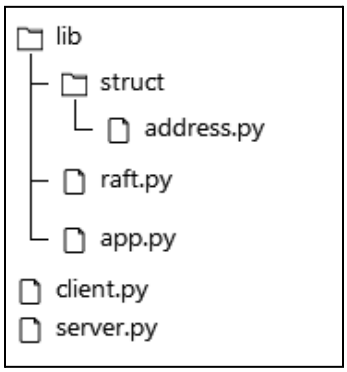
Berikut adalah contoh strategi yang dapat digunakan untuk pengimplementasian tugas besar berikut.

1. Komunikasi antar node dengan menggunakan **XMLRPC**
2. Pesan akan dienkode dalam bentuk **string JSON** agar mudah dibaca
3. Pengimplementasian algoritma Raft akan dilakukan dalam sebuah kelas utama bernama **RaftNode** yang berisi fungsionalitas
    a. **Heartbeat**
    b. **Leader Election**
    c. **Log Replication**
    d. **Task Execution**
    e. **Membership Change**
4. Identifier dari masing-masing RaftNode adalah pasangan unik IP dan port
5. Operasi yang dilakukan di dalam masing-masing node dilakukan secara multithreading agar operasi yang dilakukan lebih konsisten
6. Ketika menjalankan suatu node dengan argumen contact address kosong, maka node tersebut akan secara otomatis menjadi **Leader**. Jika tidak kosong, maka node tersebut akan mencoba untuk meng-apply membership ke contact address tersebut
7. Pengiriman heatbeat dilakukan secara berkala dengan sebuah jangka waktu konstan tertentu.
8. Seluruh **client request** dan **membership change** yang masuk ke node bukan leader akan **mengembalikan error dan address leader**; kemudian client tersebut akan mengirim ulang request tersebut ke address leader yang diterima sebelumnya
9. Proses perubahan sebuah node dari cluster dilakukan secara manual (fungsionalitas tidak diimplementasikan di dalam kelas RaftNode, namun dilakukan dengan mematikan program secara langsung) dan proses ini berjalan secara *graceful* (tidak akan merusak cluster secara keseluruhan)
10. Roadmap implementasi:
    a. Perancangan sistem cluster dan metode komunikasi yang akan digunakan
    b. Pendefinisian RPC yang dapat dilakukan oleh masing-masing node
    c. Pengimplementasian kerangka kelas RaftNode
    d. Pengimplementasian mekanisme **Heartbeat** secara berkala
    e. Pengimplementasian mekanisme **Leader Election**
    f. Pengimplementasian mekanisme **Log Replication**
    g. Pengimplementasian mekanisme **Task Execution**
    h. Pengimplementasian mekanisme **Membership Change**
    i. Pengimplementasian unit test dan proses testing

3.3. Implementasi & Contoh

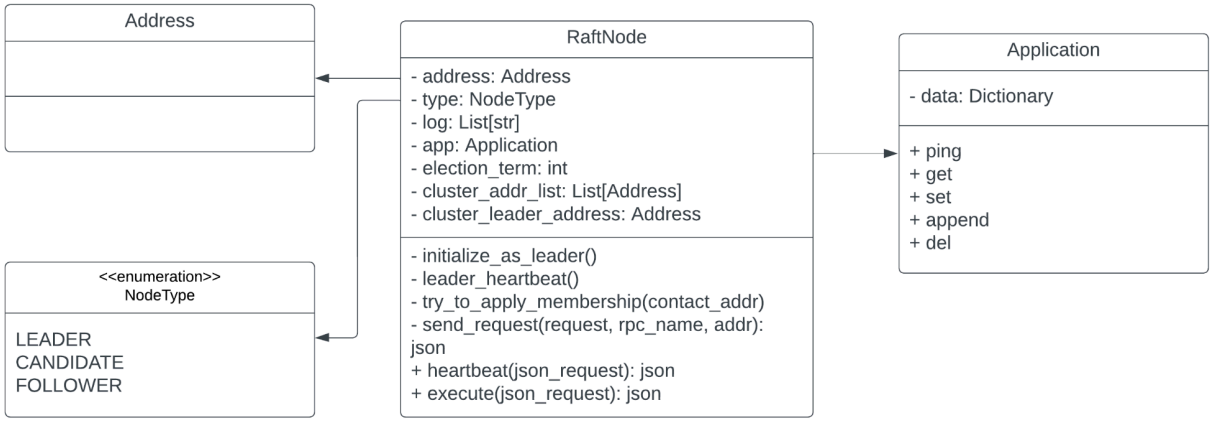
Bagian ini adalah ***contoh*** dari beberapa implementasi **Raft** yang dapat digunakan. Lanjutkan bagian ini sesuai dengan rencana yang telah dibuat. Jika merasa desain dan kode yang disediakan terlalu redundan atau tidak efisien, **bagian ini tidak wajib diikuti** dan **diperbolehkan untuk membuat semuanya dari scratch.**

Contoh Struktur Folder



Contoh struktur folder

Contoh Diagram Kelas



Contoh diagram kelas

Contoh Kode

Catatan : Kode hanya ditujukan untuk ilustrasi, ***kode mungkin bekerja dan mungkin tidak***
Perbaiki dan sesuaikan dengan keperluan masing-masing jika menggunakan contoh

address.py

```python
class Address(dict):
    def __init__(self, ip: str, port: int):
        dict.__init__(self, ip=ip, port=port)
        self.ip   = ip
        self.port = port

    def __str__(self):
        return f"{self.ip}:{self.port}"

    def __iter__(self):
        return iter((self.ip, self.port))

    def __eq__(self, other):
        return self.ip == other.ip and self.port == other.port

    def __ne__(self, other):
        return self.ip != other.ip or self.port != other.port
```

server.py

```python
from address       import Address
from raft          import RaftNode
from xmlrpc.server import SimpleXMLRPCServer
from app           import KVStore



def start_serving(addr: Address, contact_node_addr: Address):
    print(f"Starting Raft Server at {addr.ip}:{addr.port}")
    with SimpleXMLRPCServer((addr.ip, addr.port)) as server:
        server.register_introspection_functions()
        server.register_instance(RaftNode(KVStore(), addr, contact_node_addr))
        server.serve_forever()



if __name__ == "__main__":
    if len(sys.argv) < 3:
        print("Usage: server.py ip port [contact_ip] [contact_port]")
        exit()

    contact_addr = None
    if len(sys.argv) == 5:
        contact_addr = Address(sys.argv[3], int(sys.argv[4]))
    server_addr = Address(sys.argv[1], int(sys.argv[2]))
```

```
    start_serving(server_addr, contact_addr)
```

raft.py

```
import asyncio
from threading     import Thread
from xmlrpc.client import ServerProxy
from typing        import Any, List
from enum          import Enum
from address       import Address



class RaftNode:
    HEARTBEAT_INTERVAL   = 1
    ELECTION_TIMEOUT_MIN = 2
    ELECTION_TIMEOUT_MAX = 3
    RPC_TIMEOUT          = 0.5

    class NodeType(Enum):
        LEADER    = 1
        CANDIDATE = 2
        FOLLOWER  = 3

    def __init__(self, application : Any, addr: Address, contact_addr: Address = None):
        socket.setdefaulttimeout(RaftNode.RPC_TIMEOUT)
        self.address:             Address           = addr
        self.type:                RaftNode.NodeType = RaftNode.NodeType.FOLLOWER
        self.log:                 List[str, str]    = []
        self.app:                 Any               = application
        self.election_term:       int               = 0
        self.cluster_addr_list:   List[Address]     = []
        self.cluster_leader_addr: Address           = None
        if contact_addr is None:
            self.cluster_addr_list.append(self.address)
            self.__initialize_as_leader()
        else:
            self.__try_to_apply_membership(contact_addr)



    # Internal Raft Node methods
    def __print_log(self, text: str):
        print(f"[{self.address}] [{time.strftime('%H:%M:%S')}] {text}")

    def __initialize_as_leader(self):
        self.__print_log("Initialize as leader node...")
        self.cluster_leader_addr = self.address
        self.type                = RaftNode.NodeType.LEADER
        request = {
            "cluster_leader_addr": self.address
```

```
        }
        # TODO : Inform to all node this is new leader
        self.heartbeat_thread =
Thread(target=asyncio.run,args=[self.__leader_heartbeat()])
        self.heartbeat_thread.start()

    async def __leader_heartbeat(self):
        # TODO : Send periodic heartbeat
        while True:
            self.__print_log("[Leader] Sending heartbeat...")
            pass
            await asyncio.sleep(RaftNode.HEARTBEAT_INTERVAL)

    def __try_to_apply_membership(self, contact_addr: Address):
        redirected_addr = contact_addr
        response = {
            "status": "redirected",
            "address": {
                "ip":   contact_addr.ip,
                "port": contact_addr.port,
            }
        }
        while response["status"] != "success":
            redirected_addr = Address(response["address"]["ip"],
response["address"]["port"])
            response        = self.__send_request(self.address, "apply_membership",
redirected_addr)
        self.log                = response["log"]
        self.cluster_addr_list   = response["cluster_addr_list"]
        self.cluster_leader_addr = redirected_addr

    def __send_request(self, request: Any, rpc_name: str, addr: Address) -> "json":
        # Warning : This method is blocking
        node         = ServerProxy(f"http://{addr.ip}:{addr.port}")
        json_request = json.dumps(request)
        rpc_function = getattr(node, rpc_name)
        response     = json.loads(rpc_function(json_request))
        self.__print_log(response)
        return response

    # Inter-node RPCs
    def heartbeat(self, json_request: str) -> "json":
        # TODO : Implement heartbeat
        response = {
            "heartbeat_response": "ack",
            "address":            self.address,
        }
        return json.dumps(response)


    # Client RPCs
    def execute(self, json_request: str) -> "json":
        request = json.loads(json_request)
```

```
        # TODO : Implement execute
        return json.dumps(request)
```

IV. Penilaian & Bonus

Sama seperti tugas kecil dan penilaian Laboratorium Sister sebelumnya, program akan dinilai secara kelompok. Untuk nilai demo akan dinilai secara individu.

- Server dan client dapat berkomunikasi **(10)**
- Fitur pada protokol Raft berjalan dengan baik **(50)**
  - **Heartbeat** **(10)**
  - **Leader Election** **(10)**
  - **Log Replication** **(15)**
  - **Membership Change** **(15)**
- Demo **(40)**
- Bonus **(max. 20)**

Berikut adalah bonus (***yang bersifat opsional***) tersedia pada tugas besar ini. Nilai dari bonus maksimal **20**.

- **Unit Test** **(10)**
  Implementasikan unit test untuk memastikan semua komponen berfungsi dengan baik.

- **Transaction** **(10)**
  Implementasikan fitur untuk mengeksekusi beberapa perintah sebagai suatu transaksi.

- **Log Compaction** **(10)**
  Implementasikan fitur log compaction pada protokol Raft. Untuk log compaction, log harus dalam bentuk persistent storage.

V. Pengumpulan dan Deliverables

1. Pengerjaan tugas dilakukan dengan membuat sebuah repository pada Assignment di GitHub Classroom “Lab Sister 21”. Pastikan bahwa project visibility kelompok private selama pengerjaan
2. Pengerjaan tugas dilakukan berkelompok yang sama dengan kelompok tugas kecil. Berikut adalah sheet daftar kelompok IF3230 - Daftar Kelompok.
3. Apabila ada pertanyaan lebih lanjut, jangan lupa untuk selalu kunjungi sheet QnA
4. **Catatan penting:** Jika diketahui terdapat kode yang sama dengan repository di internet, **maka akan dianggap melakukan kecurangan**. Alasan menggunakan fitur kode autocomplete seperti **vibe coding** yang melakukan copas akan diabaikan.
5. **Segala kecurangan baik sengaja dan tidak disengaja akan ditindaklanjuti oleh pihak asisten, yang akan berakibat sanksi akademik ke setiap pihak yang terlibat**
6. Deadline pengerjaan tugas ini adalah **Kamis, 5 Juni 2025, 23.59 WIB**, semua commit harus dilakukan sebelum jam tersebut
7. Pengumpulan dilakukan dengan melakukan **release** pada repository Github sebelum deadline. Revisi pengumpulan dapat dilakukan dengan membuat release baru.

VI. Tata Cara Demo

Demo akan dilakukan secara luring dengan jadwal dibagikan menyusul menjelang deadline. Ketentuan pelaksanaan demo antara lain:

Setup

0. ~~Setup tidak boleh menggunakan localhost~~ **Gunakanlah beberapa komputer dalam jaringan lokal atau virtual network dengan menggunakan vm (docker terhitung sebagai virtual network jadi boleh).** Satu perangkat bisa menjadi beberapa node dengan menggunakan bridge.

1. Lakukan **`git status`** dan **`git log`** pada repository

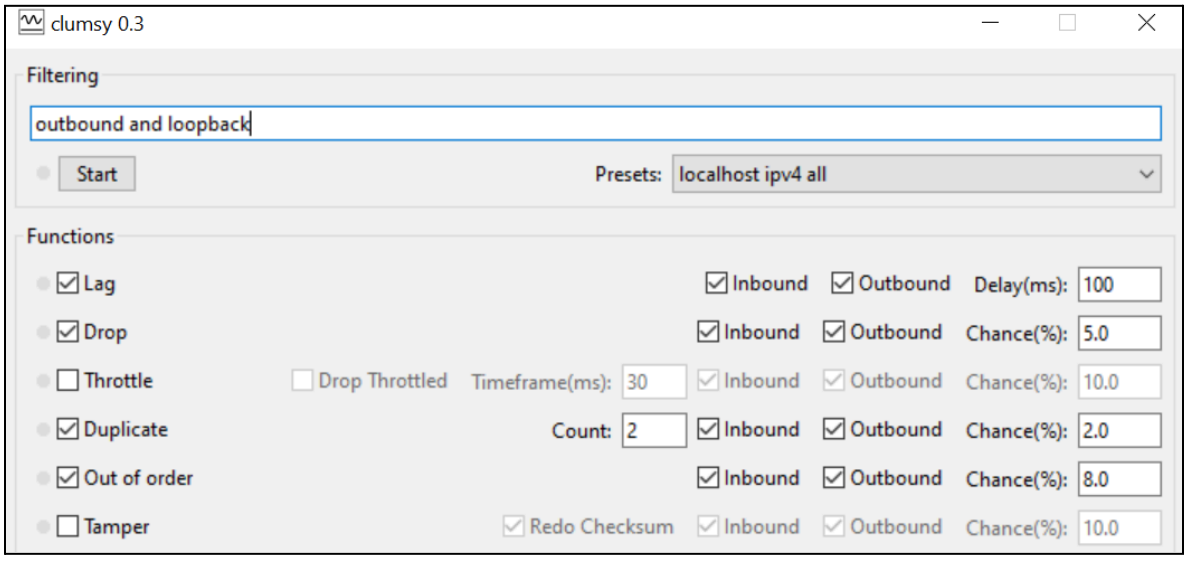
2. Jelaskan secara singkat sistem dan overview fungsi & method yang diimplementasikan

3. Gunakan `tc` command berikut (ala *IF3130 - Jaringan Komputer*)

   ```
   tc qdisc add dev lo root netem delay 100ms 50ms reorder 8% corrupt 5% duplicate 2% 5% loss 5%
   ```

   Apabila command `tc` tidak dapat dieksekusi pada vm atau docker, lakukan setup (langkah 0.) pada localhost dengan konfigurasi clumsy (langkah 4.)

4. Jika ingin menggunakan Windows untuk demo, subtitusi command tc dengan github.com/clumsy. Gunakan konfigurasi berikut

5. Jika diperlukan, diperbolehkan juga untuk mengubah konfigurasi ketika menjalankan environment demo. Pastikan untuk memperlihatkan pada awal hanya mengubah konfigurasi, **bukan kode utama**

6. Jika ada behavior yang unexpected karena network, jelaskan juga sembari memperbaiki jika diperlukan

7. Gunakan 4 node berbeda untuk server. Client berasal dari luar keempat node tersebut.

Demo Eksekusi Program (40)

Penilaian akan mempertimbangkan keberhasilan eksekusi program, progress pengerjaan, dan penjelasan dari implementasi.

Heartbeat (10)

1. Tampilkan **Heartbeat** pada setiap server (Tunjukkan pula node mana yang menjadi **Leader**)

2. Untuk memastikan koneksi sudah terhubung, kirimkan client request ping ke salah satu node

Log Replication (10)

1. Kirimkan client request berikut ke **Leader** hingga tereksekusi
    a. `set("1", "A")`
    b. `append("1", "BC")`
    c. `set("2", "SI")`
    d. `append("2", "S")`
    e. `get("1")`

2. Kirimkan client request berikut ke **node bukan Leader** hingga tereksekusi
    a. `get("1")`
    b. `get("2")`

3. Kirimkan client request berikut dari **dua node ke Leader secara hampir bersamaan** hingga tereksekusi
    Node 1:
    a. `set("ruby-chan", "choco-minto")`
    b. `append("ruby-chan", "-yori-mo-anata")`

    Node 2:
    a. `set("ayumu-chan", "strawberry-flavor")`
    b. `append("ayumu-chan", "-yori-mo-anata")`

4. Kirimkan client request berikut ke **node bukan Leader yang berbeda** hingga tereksekusi
    a. `get("ruby-chan")`
    b. `get("ayumu-chan")`

Leader Election (10)

1. Setelah dieksekusi, **matikan** (dengan CTRL+C misalnya) **Leader**
2. Tunggu dan perlihatkan proses **Leader Election** pada 3 node sisa
3. Kirimkan client request berikut ke **node Leader yang baru**
    a. `strlen("1")`
    b. `strlen("2")`
    c. `del("1")`
    d. `append("2", "TE")`
    e. `append("2", "R")`
4. Nyalakan kembali **node yang mati sebelumnya menggunakan address yang sama dan mendaftarkan diri ke Leader yang baru terpilih**
5. Kirimkan client request berikut ke **Leader** hingga tereksekusi
    a. `set("3", "")`
    b. `append("3", "UwU")`
    c. `append("4", "Onii-Chan")`
    d. `append("4", "Daisuki")`
6. Kirimkan client request berikut ke **node yang baru saja hidup** hingga tereksekusi
    e. `get("2")`
    f. `get("3")`
    g. `get("4")`
7. Kirim request berikut ke **node Leader** dan tampilkan ke layar
`request_log()`

Membership Change (10)

1. **Tambah satu node baru** sebagai member cluster. Tunjukkan proses membership change dan replikasi log ke node yang baru ditambahkan, tunggu sampai proses tersebut selesai.
2. Kirim request berikut ke **node yang baru ditambahkan** dan tampilkan ke layar `request_log()`. Setelah mendapatkan alamat leader, kirim `request_log()`ke leader.
3. **Lakukan dua hal berikut secara bersamaan:**
    a. **Hapus salah satu node node dan tambahkan node yang lain** sebagai member cluster.
    b. Kirim request berikut ke **Leader**
    `set("crocodilo", "bombardino") → set("tung-tung-tung", "sahur")`

*Maksudnya di sini adalah mengirim request tanpa menunggu node baru untuk selesai sinkronisasi log

4. Kirim request berikut ke **node yang baru ditambahkan** dan tampilkan ke layar `request_log()`. Setelah mendapatkan alamat leader, kirim `request_log()` ke leader.

Closing

Jangan lupa untuk menggunakan command berikut untuk revert konfigurasi `qdisc` pada `tc`

```
tc qdisc del dev lo root netem delay 100ms 50ms reorder 8% corrupt 5% duplicate 2% 5% loss 5%
```

VII. Referensi

1. https://raft.github.io/ - Raft Paper - Github Pages
2. https://raft.github.io/raft.pdf - Original Raft Paper
3. http://thesecretlivesofdata.com/raft/ - Raft visualization
4. https://www.wikiwand.com/en/Raft_(algorithm) - Raft Wikipedia
5. https://www.cl.cam.ac.uk/teaching/2122/ConcDisSys/dist-sys-notes.pdf
6. Membership Change: https://github.com/hashicorp/raft/blob/main/membership.md

~ The name Raft implies the existence of a greater (water vessel) ~
Johann

~ ~
yellow

~ Terakhiran? ~
Awe

~ 空を越えて さあ舞い上がれ ~
Duke

~ last kah ~
Willy

~ "Where's SISTER 2, William? You were supposed to drop him off an hour ago. 😡" ~
yujin

~ "Patrick, aku kira Raft bukan kata yang benar."
"AYOLAHH kau tau? Aku Raft kau Raft dia Paxos... Aku Raft Raft Rafting Raftou Raftologi? Ingin belajar Raft? Itu tingkat pertama, Spongebob!" ~
Toper

~ it would be embarrassing when we meet again ~
barkod