

Tugas Besar 1

IF3130 - Sistem Paralel dan Terdistribusi

“The Final Chapter”

Consensus Protocol: Raft

Dipersiapkan oleh
Asisten Lab Sistem Terdistribusi



Waktu Mulai

Senin, 17 November 2025, 21:00 WIB

Waktu Akhir

~~Jumat, 12 Desember 2025, 23.59 WIB~~

Jumat, 19 Desember 2025, 23.59 WIB (wajib selesai)

Changelog

1. Revisi terkait platform pelaksanaan demo (demo dilakukan daring secara *default*).
2. Perpanjangan tenggat akhir tugas besar menjadi h+7 (Jumat, 19 Desember 2025, 23.59).

I. Latar Belakang

Anda disarankan untuk membaca bagian ini.

TVTropes – the "Title Drop Chapter"

"Sometimes, a chapter in a book, or an episode in a TV series, or a part in another type of work, shares its title with the work itself.

This often happens with the episode or part. One of the reasons is that the title works as indication that the road is now complete, for example, if an essay's title refers to the main idea it's trying to prove, the last chapter would have the same title because all the arguments have been presented and now the conclusion represents the whole book, similarly to a fractal, where a part is identical to the whole."

<=====0
=====>

Dengan perlahan, Kamu menghampiri bukit. Hawa-hawa tidak enak mulai menggusur bahumu. Sejauh ini, kamu sudah melalui banyak sekali rintangan, kamu pun bingung kapan berakhirnya semua kesulitan ini.

Kamu mengusap keringatmu di leher. Tidak lama kemudian, kamu melihat sebuah sosok yang familier, tetapi terasa aneh.



?????

Permisi...?

Kamu ucapan.

“ ”

“ ”

Tidak ada jawaban.

Kamu menunggu beberapa saat.

Sosok tersebut masih diam.

Kamu pun mencoba bertanya lagi.

Halo...? Apakah aku sudah berhasil menyelesaikan semua rintangan-

Belum.

Kamu belum selesai.

Sosok ini memotong Kamu sebelum menyelesaikan kalimat tersebut. Dia menjawab tanpa membalikkan badannya.

Kamu terdiam sejenak.

...

Ya sudah, lantas kapan akan berakhirnya perjalananku ini?!

Tenang saja, perjalanamu akan kuakhiri di sini.

Namun, sebelum itu, aku harus mengujimu untuk mengetahui apakah kamu memang cocok menjadi lawanku.

Jantungmu mulai berdetak lebih cepat. Kamu sama sekali gak tahu apa yang akan terjadi. Apa yang dimaksud perjalanamu akan diakhiri di sini? Kamu hanya bisa terdiam dan menunggu.

Pertama, berapakah 13522420 DAN -1?

Dalam hati kamu berkata, apa maksudnya “berapa 13522420 dan -1?” Itu bahkan tidak masuk akal!

Namun, kamu mengingat kembali masa-masamu praktikum bitwise di **IF1230 Organisasi dan Arsitektur Komputer**. Kemudian kamu dengan ragu mencoba menjawab.

“Apakah jawabannya 13522420? karena N & 0xFFFFFFFF menghasilkan N itu sendiri.”

Jawabanmu benar.

Kedua, apabila nilai yang tersimpan dalam rdx bernilai 0xFFFFFFFF87cc26ab, berapakah nilai yang tersimpan dalam nilai yang tersimpan dalam edx?

Dalam hati kamu bingung, apa kamu tidak salah dengar, kata-kata “nilai yang tersimpan” diulang sekali? Kamu pun ingat di IF1230 kamu mempelajari apa maksud nilai yang tersimpan dalam pointer. Dengan begitu kamu bisa menjawab,

Ya tentu tidak diketahui, selama kita tidak tahu apa yang tersimpan dalam 0x87cc26ab.

Baik.

Berikutnya, berapa kali kode berikut ini melakukan fork?

```
#include <stdio.h>
#include <unistd.h>
#include <stdlib.h>

int main() {
    int *p = malloc(sizeof(int));
    if (fork()) *p = fork();
    else free(p);
    if (fork() && *p) fork();
    printf("pid: %d, val: %d\n", getpid(), *p);
}
```

Kamu mulai paham maksud dari ujian ini. Dengan pengetahuanmu di **IF2130 Sistem Operasi**, kamu langsung menjawab,

Tidak bisa diketahui. Kode yang kau berikan mengandung undefined behavior karena kemungkinan melakukan dereference pada null pointer. Dari definisi undefined behavior, kode itu bisa melakukan berapa pun kali fork setelah itu.

Hmm. Ya sudah. Ini yang terakhir.

Aku mempunyai IP address 192.168.0.56. Lakukanlah PING ke device-ku.

Kamu sudah lulus IF2230 Jaringan Komputer. Kamu bukan lagi bocil yang bisa ditipu pertanyaan selevel itu.

Tidak bisa. Aku butuh public IP-mu untuk melakukan itu.

Sosok itu pun tersenyum, mengeluarkan tawa kecil. Kemudian akhirnya, dia membalikkan badannya untuk memandangmu, seolah-olah sudah menganggapmu setara.

Hihahi. Jadi begitu rupanya.



[Lv. 67 / FURANI – FONTAENI]

"Aku sudah paham. Kalau dengan kata-kata Tai Lung dari Kung Fu Panda: finally, a worthy opponent, our battle will be legendary!"

"Sekarang, sudah waktunya. Mendekatlah kau, dengan IF3130 Sistem Parallel dan Terdistribusi, aku akan mengajarkan apa artinya menjadi mahasiswa IF yang sebenarnya."

New objective: Kerjakan tubes IF3130 and defeat Furani Fontaeni!

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, tRPC, 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 **strIn** digunakan untuk mendapatkan panjang value dari *key* yang diberikan

```
> strIn <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.

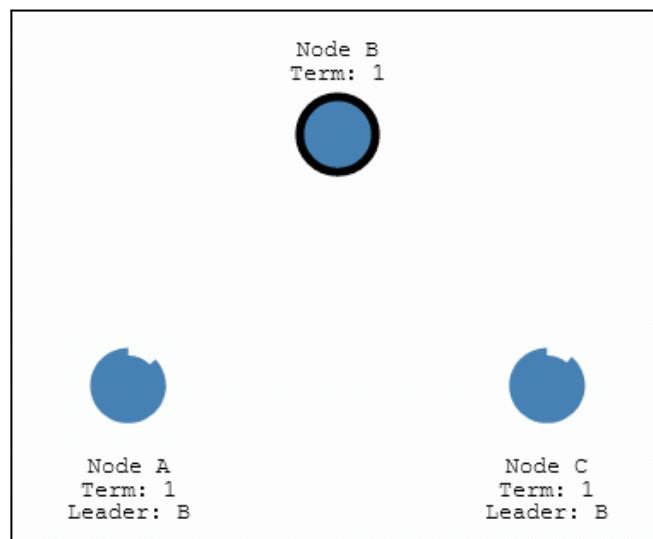
Good Luck, Have Fun ;)

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. Namun, jika Anda ingin Raft versi *pure* dan *unfiltered*-nya, silakan langsung kunjungi paper untuk Raft yang sudah tersedia di tautan Github Pages yang disediakan.



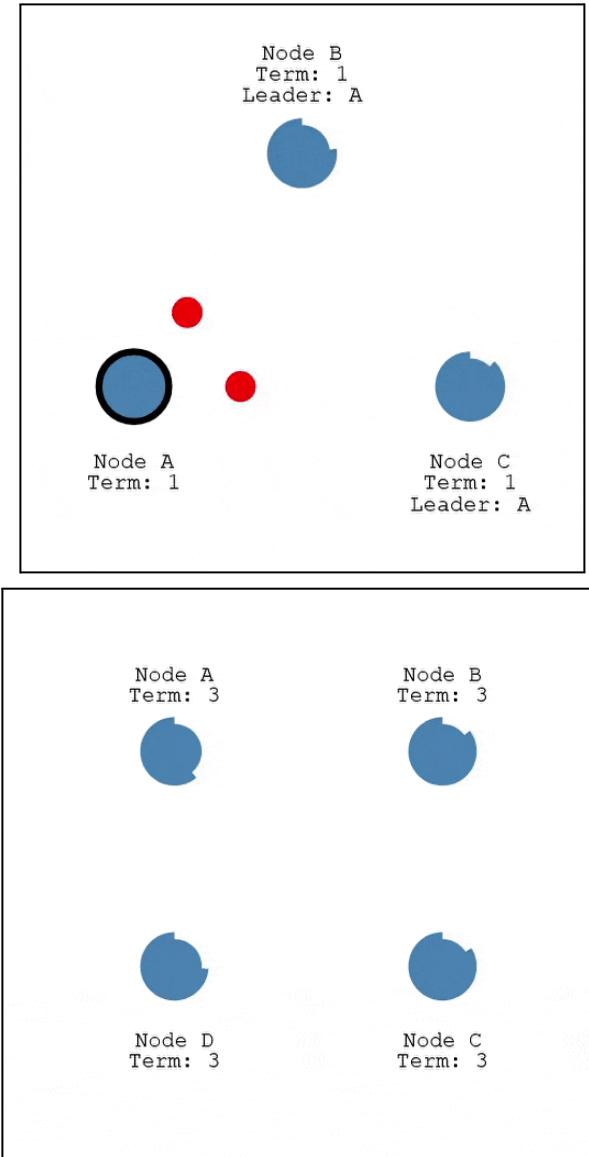
Raft Visualization - Heartbeat

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 ini merupakan salah satu strategi yang dapat dijadikan acuan dalam melaksanakan tugas besar berikut.

(note: ini hanya sebagai gambaran, penerapan dapat disesuaikan dengan keyakinan masing-masing.)

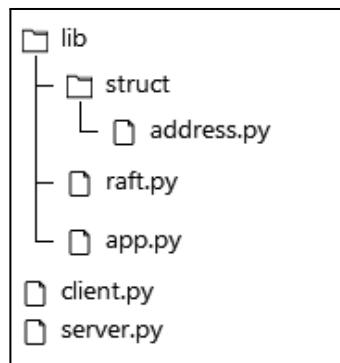
1. Komunikasi antar node dilakukan melalui **XMLRPC**.
2. Pesan yang dipertukarkan akan dikodekan menggunakan format **JSON string** untuk mempermudah proses pembacaan.
3. Penerapan algoritma Raft akan dilakukan dalam sebuah kelas utama bernama **RaftNode** yang berisi berbagai fungsi, yaitu:
 - a. **Heartbeat**
 - b. **Leader Election**
 - c. **Log Replication**
 - d. **Task Execution**
 - e. **Membership Change**
4. Setiap RaftNode memiliki **identitas unik** berupa kombinasi IP dan port yang terbagi atas tiga role **leader**, **follower**, **leader-candidate** (follower yang mencalonkan diri sebagai leader, masih nunggu voting).
5. Aktivitas dalam setiap node dijalankan menggunakan **multithreading**, sehingga proses yang berlangsung dapat lebih **stabil dan sinkron**.
6. Ketika sebuah node dijalankan dengan **contact address kosong**, maka node tersebut akan otomatis berperan sebagai **Leader**. Namun jika contact address diberikan, node tersebut akan berusaha melakukan **membership apply** ke alamat tersebut.
7. Pengiriman **heartbeat** dilakukan secara rutin dalam interval waktu yang **tetap** oleh **leader** ke **follower**. Apabila, pada rentang waktu tertentu (timeout yang ditentukan) **follower** tidak menerima *heartbeat* dari **leader**, terjadi proses **Leader Election**.
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 **penghapusan atau keluarnya** sebuah node dari cluster dilakukan secara manual. Hal ini tidak ditangani oleh fungsi di dalam kelas RaftNode, melainkan dengan cara menghentikan program secara langsung. Proses ini tetap berlangsung secara *graceful*, sehingga tidak menyebabkan kerusakan pada keseluruhan cluster.
10. Roadmap implementasi:
 - a. Perancangan struktur cluster beserta metode komunikasinya
 - b. Penentuan RPC yang akan didukung oleh setiap node
 - c. Pembuatan kerangka kelas RaftNode

- d. Implementasi mekanisme Heartbeat secara periodik
- e. Implementasi mekanisme Leader Election
- f. Implementasi mekanisme Log Replication
- g. Implementasi mekanisme Task Execution
- h. Implementasi mekanisme Membership Change
- i. Penulisan unit test serta proses pengujian sistem

3.3. Implementasi & Contoh

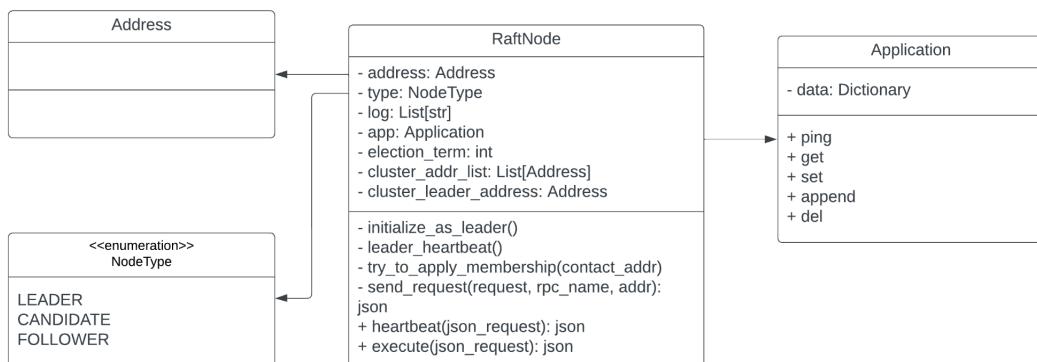
Bagian ini adalah ***contoh*** dari beberapa implementasi **Raft** yang dapat digunakan. Lanjutkan bagian ini sesuai dengan rencana yang telah dibuat. Jika merasa desain dan kode yang disediakan terlalu redundan atau tidak efisien, **bagian ini tidak wajib diikuti dan diperbolehkan untuk membuat semuanya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

```
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

```
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. Pengerajan tugas dilakukan dengan membuat sebuah repository pada [Assignment di GitHub Classroom “Lab Sister 22”](#). Pastikan bahwa project visibility kelompok private selama pengerajan
2. Pengerajan 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 pengerajan tugas ini adalah **Jumat, 12 Desember 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/daring/luring, tergantung kesepakatan asisten dan kelompok, dengan pilihan default secara daring/online) dengan pemilihan jadwal dibagikan menyusul mendekati/setelah deadline. Ketentuan pelaksanaan demo antara lain:

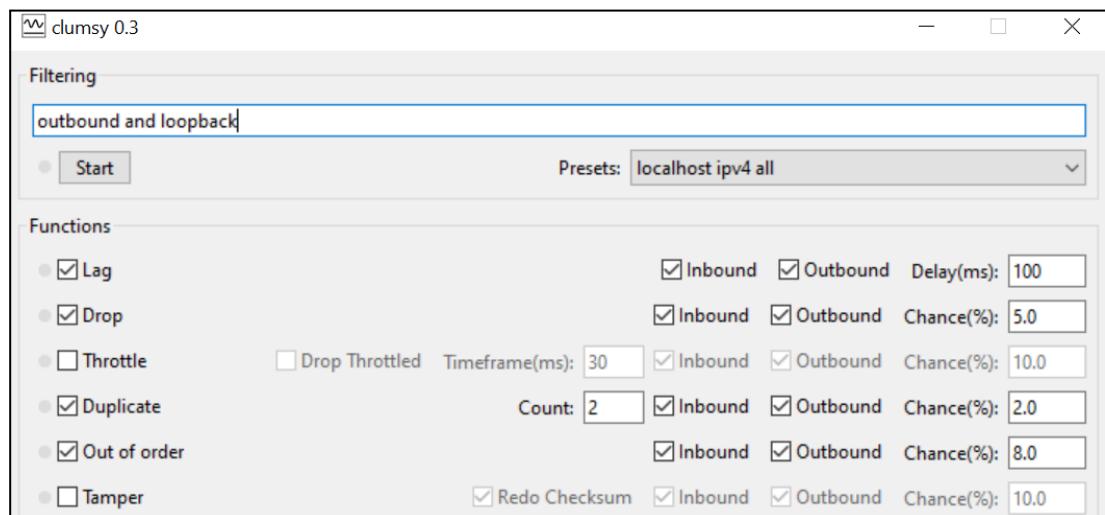
Setup

0. 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 *IF2230 - 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, substitusi command `tc` dengan github.com/clumsy. Gunakan konfigurasi berikut



5. Pastikan ada **delay** pada antar node sebelum memulai demo, harap untuk **menunjukkannya** (bisa melalui beberapa *command*, seperti ping dsb) sebelum demo dimulai. Hal seperti lupa untuk menerapkan delay atau bahkan tidak bisa akan berdampak **besar pada penilaian**.
6. Jika diperlukan, diperbolehkan juga untuk mengubah konfigurasi ketika menjalankan **environment** demo. Pastikan untuk memperlihatkan pada awal hanya mengubah konfigurasi, **bukan kode utama**
7. Jika ada behavior yang unexpected karena network, jelaskan juga sembari memperbaiki jika diperlukan
8. Gunakan 4 node berbeda untuk server. Client berasal dari luar keempat node tersebut.

Demo Eksekusi Program (40)

Penilaian akan mempertimbangkan keberhasilan eksekusi program, progress penggeraan, 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** dan **tambahkan node yang lain** sebagai member cluster.
 - b. Kirim request berikut ke **Leader**
`set("crocodillo", "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_\(algorithm\)](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>



~ apakah akan ada tubes 2, hmm.... ~ Ali



~ Raft ~ Indra