Tugas Besar 1
IF3230 - Sistem Paralel dan Terdistribusi
何これ

Consensus Protocol: Raft

Dipersiapkan oleh
Asisten Lab Sistem Terdistribusi

Waktu Mulai

Sabtu, 8 April 2023, 18.00 WIB

Waktu Akhir

Senin, 22 Mei 2023, 23.59 WIB

I. Latar Belakang

Tinggal di desa ninja yang terkenal dengan perguruan tinggi ninja terbaik di dunia bukanlah sesuatu yang mudah, Rian adalah seorang mahasiswa teknik informatika yang sedang belajar sistem paralel dan terdistribusi. Ia telah melewati banyak hal selama masa studi, dari begadang di restoran ayam cepat saji Mukidi (MkD) hingga minum kopi instan dan memakan indomie sepanjang hari.

Rian 👍

Kehidupan Rian cukup sibuk dengan jadwal kuliah yang padat dan latihan ninja di pagi hari. Namun, dia memutuskan untuk ~~menambah beban pikirannya dengan~~ belajar sistem paralel dan terdistribusi. Dia selalu berusaha keras untuk mengikuti kuliah dan belajar mandiri dengan bantuan internet dan buku-buku. Namun, semakin dia mempelajari topik tersebut, semakin rumit dan sulit. Seringkali, Rian merasa stres karena tugas-tugas yang sulit dan ujian yang akan datang. Namun, dia selalu menemukan cara untuk melampiaskan stresnya dengan berlatih senjata ninja atau menonton anime kesukaannya. Dia juga menemukan teman-teman yang menyenangkan di kelas dan mereka sering membantunya dalam belajar.

Suatu hari, Rian dan teman-temannya sedang belajar bersama di kampus, tiba-tiba mereka diserang oleh sekelompok ninja jahat. Rian dan teman-temannya berusaha bertarung dengan keahlian ninja mereka, namun mereka kalah jumlah. Ketika mereka hampir putus asa, tiba-tiba seorang guru ninja muncul dan membantu mereka mengalahkan musuh-musuh mereka. Rian dan teman-temannya sangat bersyukur dan mereka memperkenalkan diri kepada guru ninja tersebut.

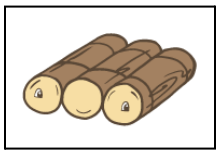
Profesor Ninja 👍

Ternyata, guru ninja tersebut adalah seorang profesor di jurusan informatika yang mengajar sistem paralel dan terdistribusi! Rian dan teman-temannya tidak pernah menyangka bahwa profesor mereka adalah seorang ninja hebat. Profesor tersebut memberikan saran dan bantuan dalam belajar dan mengajar mereka teknik ninja baru yang berguna dalam dunia informatika, seperti rasengan, menangkal serangan kuntilanak, beternak babi ngepet berekor 9, dll.

Dengan bantuan profesor ninja tersebut, Rian semakin termotivasi untuk belajar dan meraih keberhasilan dalam kuliahnya. Dia memutuskan untuk berlatih lebih keras untuk menjadi seorang ninja yang lebih baik dan mempelajari sistem paralel dan terdistribusi dengan lebih baik lagi. Untuk meningkatkan pengetahuannya akan sistem paralel dan terdistribusi, bantu Rian dalam mengerjakan tugas berikut

II. Spesifikasi Tugas

Tugas besar ini akan menugaskan untuk membuat sebuah 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

Berbeda dengan tugas-tugas sebelumnya, spesifikasi tugas besar ini hanya akan meminta interface-interface dan *requirement* sistem yang harus dipenuhi

Desain sistem dan detail implementasi akan sepenuhnya diserahkan kepada praktikan

Berikut adalah spesifikasi dan requirement dari tugas besar:

1. Bahasa yang digunakan adalah **Python** atau **Rust** (Bonus)
2. **Hanya built-in library dan beberapa library "external" diperbolehkan untuk Python**
    a. Library external adalah library yang tidak ada pada built-in Python library
    b. Library external yang diperbolehkan adalah **library networking** (**HTTP**, **gRPC**, etc)
    c. Pastikan library external tertulis di dalam **requirements.txt**
    d. Batasan untuk **Rust** dapat dilihat pada bagian Penilaian & Bonus
    e. Jika dibutuhkan library external lain, dapat ditanyakan pada sheets QnA
3. Program adalah *distributed message queue* dengan layanan **enqueue** dan **dequeue**
    a. **Tipe data** untuk message yang disimpan pada *queue* adalah sebuah **string**
    b. Hasil **dequeue** adalah sebuah **string** dan **enqueue** dilakukan dengan mengirim sebuah **string**. Asumsikan tidak pernah melakukan **dequeue** kosong
4. Cluster server dapat menjaga **Consistency** dan **Availability**
5. Implementasi Raft harus menyediakan
    a. **Membership Change** (Mechanism for adding another server node)
    b. **Log Replication** (Cluster action logging system)
    c. **Heartbeat** (Node health monitoring & periodic messages)
    d. **Leader Election** (Leader node failover mechanism)
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 (enqueue/dequeue)
    b. **request_log** untuk mengembalikan log yang dimiliki **Leader**
7. **Semua aksi server wajib dilakukan logging ke terminal** (minimal deskripsikan aksi yang dilakukan node)
8. **Minimum 50% Node (Rounded up) + 1 harus menjawab ACK** sebelum request client dieksekusi
9. Node yang mati tidak secara otomatis dikeluarkan dari cluster
10. Implementasi client dibebaskan: Bahasa boleh berbeda (tidak termasuk bonus Rust), menggunakan REPL atau hardcoded satu-satu sesuai dengan ketentuan demo

Good Luck, Have Fun ;)

III. Tips Pengerjaan

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

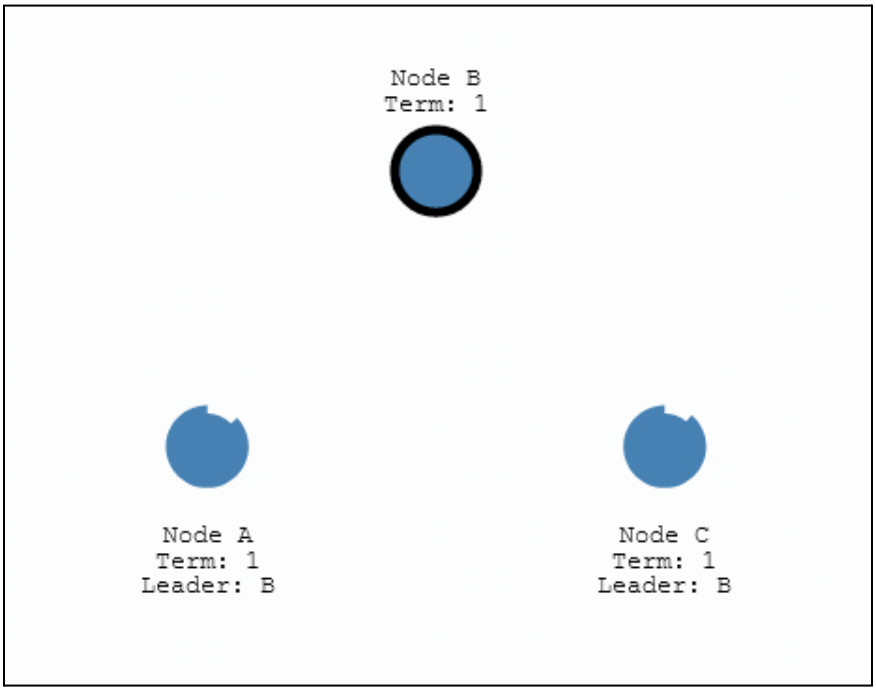
3.0. Overview - Raft Protocol

Sebelum memulai pengerjaan tugas besar ini, ada baiknya untuk memahami desain sistem yang akan diimplementasikan. Link berikut adalah webpage yang dibuat oleh author dari pembuat paper Raft. Halaman tersebut juga mencantumkan paper dalam bentuk pdf

GitHub Pages dari Author Raft : https://raft.github.io/

Selain itu, cek halaman berikut untuk *high-level overview* dan visualisasi singkat untuk Raft

Visualisasi Protokol Raft : http://thesecretlivesofdata.com/raft/



Raft Visualization - Heartbeat

3.1. Desain Sistem & Strategi

Sama ketika ingin membuat sebuah software baru dari *scratch*, pembuatan sistem akan dimulai dari desain sistem yang akan diimplementasikan. Pada tahap ini, lakukan *brainstorming* tentang requirement-requirement yang ada, contohnya adalah

- Apa metode komunikasi yang nantinya akan digunakan?
    - Apa kelebihan dan kekurangan dari setiap metode?
    - Apakah membutuhkan **blocking** / **non-blocking I/O**?
- Bagaimana API untuk pertukaran pesan antara server/client MQ dan antar-node pada cluster Raft?
- Apa format pesan yang digunakan? Binary? JSON? String?
- Apa saja fitur yang harus ada? Dan bagaimana rencana implementasinya?
    - Membership change?
    - Heartbeat?
- Apakah nantinya akan menggunakan **multithreading** / **multiprocessing**?

Tentunya karena keterbatasan pengalaman mengimplementasikan sistem seperti Raft akan mengurangi akurasi dan seberapa detail yang dapat dibahas dalam tahap *brainstorming* untuk menyerang permasalahan. **But everyone has to start somewhere**

Kumpulkan semua pertanyaan yang muncul dari setiap anggota dan diskusikan setiap approach yang akan diambil, kekurangan dan kelebihan, konsekuensi, dan sebagainya. Jika tidak ada anggota yang memiliki ide untuk meng-approach problem tertentu, problem tersebut dapat di-*defer* hingga ketika implementasi, **tetapi hindari hal tersebut sebisa mungkin**. Anggap hal tersebut sebagai "**technical debt**"

Semakin banyak pengalaman dari implementasi, semakin *akurat* dan ***powerful*** tools yang dimiliki ketika tahap *brainstorming*. Kesalahan pada tahap desain sistem adalah ***ruinously expensive mistake*** dibandingkan kesalahan seperti bug pada tahap implementasi (ala RPL)

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***

3.2. Contoh Desain & Strategi

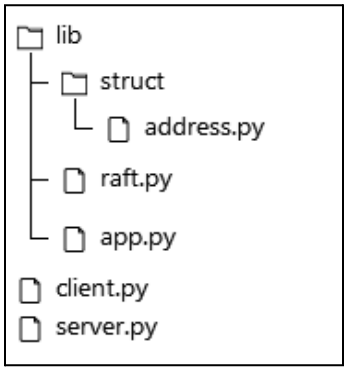
Berikut adalah contoh strategi yang digunakan untuk implementasi Python

1. Komunikasi menggunakan **RPC**, lebih akuratnya built-in **XMLRPC** pada Python
2. Pesan menggunakan **string json** yang dikirimkan lewat **RPC**
3. Akan dibuat class **RaftNode** yang memiliki atribut **Application**
4. **RaftNode** mengimplementasikan fungsionalitas
    a. **Membership Change**
    b. **Leader Election**
    c. **Log Replication**
    d. **Execute**
5. Library yang digunakan
    a. **xmlrpc** (**ServerProxy** untuk client, **SimpleXMLRPCServer** untuk server)
    b. **json** (Parser pesan)
    c. **asyncio** (Untuk asynchronous RPC)
    d. **time** (Digunakan untuk kepentingan logging)
    e. **socket** (Timeout untuk semua RPC)
    f. **threading** (Background thread)
    g. **sys** (Hanya untuk argumen)
6. **Address (IP dan port)** dari setiap node akan menjadi identifier
7. Ketika menjalankan server, jika argumen contact address kosong, secara otomatis server tersebut menjadi **Leader**. Jika tidak kosong, akan mencoba meng-**apply membership** ke cluster leader
8. Remove node diakses melalui interface untuk client **remove_server_from_list**
9. Seluruh **client request** dan **membership change** yang masuk ke node bukan Leader akan **mengembalikan error dan address Leader**. Client yang akan mencoba mengirim ulang request, bukan request diredirect oleh node **Follower**
10. Roadmap implementasi
    a. Implementasi awal adalah **server-client dengan RPC sederhana**
    b. Mengimplementasikan **Membership Change**
    c. Mengimplementasikan **Log Replication**
    d. Mengimplementasikan **Heartbeat**
    e. Mengimplementasikan **Leader Election**

3.3. Implementasi & Contoh

Dari desain dan strategi yang telah dibuat, dapat dilanjutkan pada tahap implementasi. Lanjutkan bagian ini sesuai dengan rencana yang telah dibuat.

Berikut adalah contoh struktur file yang dibuat

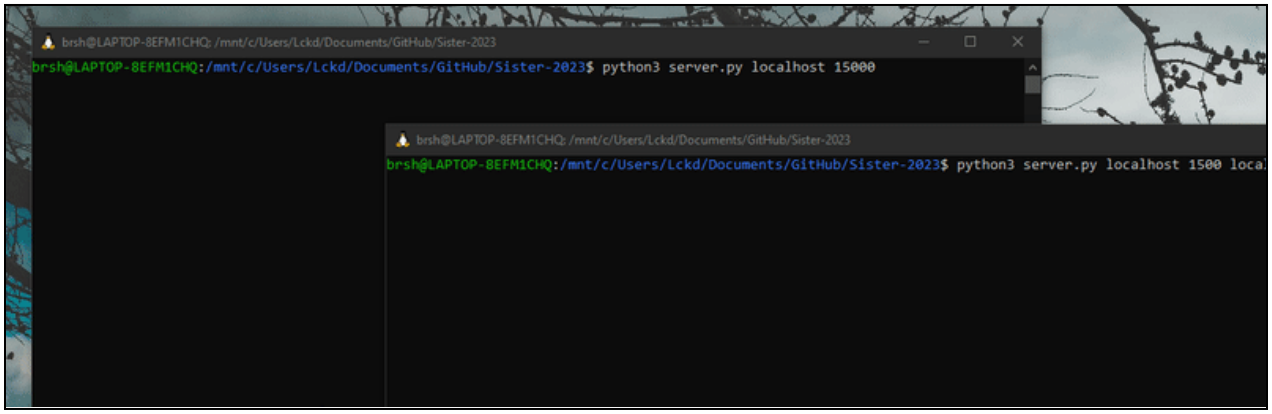


Contoh struktur folder

Halaman selanjutnya adalah ***contoh*** dari beberapa implementasi **Raft** yang dapat digunakan

Jika merasa kode terlalu redundan atau tidak efisien, sekali lagi, bagian ini tidak wajib diikuti, diperbolehkan untuk membuat semuanya dari scratch

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



Contoh eksekusi - **Menjalankan dua node, Leader port 15000 dan Follower port 1500**

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 MessageQueue



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(MessageQueue(), addr, contact_node_addr))
        server.serve_forever()



if __name__ == "__main__":
    if len(sys.argv) < 3:
        print("server.py <ip> <port> <opt: contact ip> <opt: 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. Form peer assessment dan pengumpulan demo akan ada pada bagian Pengumpulan dan Deliverables

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

Berikut adalah bonus (***yang bersifat opsional***) tersedia pada tugas besar ini

- **Implementasi menggunakan Rust (10)**
  Bonus ini meminta untuk mengimplementasikan tugas besar dengan bahasa **Rust**. Untuk mempermudah implementasi diperbolehkan untuk menggunakan library berikut: **openraft, axum/tonic/actix, tokio/async_std, reqwest, serde & turunannya, async_trait, bytes, env_logger, log, ctrlc**. Pastikan library yang digunakan tertulis pada **cargo.toml**. Jika ada library lain yang ingin digunakan dapat ditanyakan pada sheet QnA

- **Management Dashboard (10)**
  Buatlah **management dashboard untuk memonitor semua node**. Minimal dashboard dapat menampilkan isi log dan kondisi setiap node yang terhubung

- **Local Network / Internet Demo (10)**
  Tes implementasi pada **local network dengan beberapa komputer** atau **menggunakan cloud hosting**

- **Speedrun any% (20)**
  Selesaikan **tugas besar dan demo** sebelum tanggal **21 April 2023, 23.59**

- **Absurd 100% dan multi-cluster sharding (?) (very special bonus, *don't try this at home*)**
  Sama seperti **speedun any%**, tetapi **selesaikan semua bonus, ganti MQ dengan DB / KV-store dan implementasikan sharding** sebelum tanggal **14 April 2023, 23.59** dan lakukan deploy pada multi-cluster

V. Pengumpulan dan Deliverables

1. Pengerjaan tugas dilakukan dengan membuat sebuah repository pada Assignment di GitHub Classroom “Lab Sister 20”. Pastikan bahwa project visibility kelompok private selama pengerjaan
2. Pengerjaan tugas dilakukan berkelompok dengan masing-masing kelompok maksimal 5, sama seperti kelompok tugas kecil sebelumnya. Berikut adalah sheet daftar kelompok IF3230 - Daftar Kelompok
3. Apabila ada pertanyaan lebih lanjut, jangan lupa untuk selalu kunjungi sheet QnA
4. Asistensi masih ada dan dapat digunakan untuk menanyakan spesifikasi dan kendala. Berikut adalah link form Form Asistensi
5. **Catatan penting:** Jika diketahui terdapat kode yang sama dengan repository di internet, **maka akan dianggap melakukan kecurangan**. Alasan menggunakan fitur kode autocomplete seperti Github Copilot yang melakukan copas akan diabaikan
6. **Segala kecurangan baik sengaja dan tidak disengaja akan ditindaklanjuti oleh pihak asisten, yang akan berakibat sanksi akademik ke setiap pihak yang terlibat**
7. Deadline pengerjaan tugas ini adalah **Senin, 22 Mei 2023, 23.59 WIB**, semua commit harus dilakukan sebelum jam tersebut
8. Untuk demo tugas besar akan sepenuhnya menggunakan asinkron. Video demo **maksimal 30 menit**

   Deadline pengumpulan demo: **Rabu, 24 Mei 2023, 23.59 WIB**
   Link pengumpulan demo : Form pengumpulan demo tugas besar IF3230
   Link peer assessment : Form peer assessment kelompok IF3230

   Note: Untuk kelompok yang berubah pada tugas besar, isi peer assessment sesuai dengan **anggota kelompok saat tugas besar**

VI. Tata Cara Demo

Berikut adalah tata cara untuk pembuatan demo:

0. **Pastikan rekaman memperlihatkan siapa yang berbicara (jika perlu nyalakan kamera) dan setiap anggota perkenalkan diri (nama dan NIM)**
1. Lakukan **`git status`** dan **`git log`** pada repository
2. Jelaskan secara singkat sistem dan overview fungsi & method yang diimplementasikan

Demo Eksekusi Program

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

```
tc qdisc add dev lo root netem delay 1000ms 50ms reorder 8% corrupt 5%
duplicate 2% 5% loss 5%
```

2. Jika ada behavior yang unexpected karena network, jelaskan juga sembari memperbaiki jika diperlukan
3. Buka 6 terminal berbeda untuk server dan 1 untuk client
4. Nyalakan cluster server pada localhost, alokasikan address secara bebas
5. Tampilkan **Heartbeat** pada setiap terminal server
6. Kirimkan client request berikut ke **node yang bukan Leader** hingga tereksekusi
   `queue("1") → queue("2") → queue("3") → queue("s") → queue("i") → queue("s")`
7. Setelah dieksekusi, **matikan** (dengan CTRL+C misalnya) **Leader dan 1 node lain**
8. Tunggu dan perlihatkan proses **Leader Election** pada 4 node sisa
9. Kirimkan client request berikut ke **node Leader yang baru**
   `dequeue() → dequeue() → dequeue() → queue("t") → queue("e") → queue("r")`
10. Nyalakan kembali **2 node yang mati sebelumnya menggunakan address yang sama dan mendaftarkan diri ke Leader yang baru terpilih**
11. Kirimkan client request berikut ke **node yang baru saja hidup** hingga tereksekusi
    `queue("U") → queue("w") → queue("U")`
12. Terakhir, kirim request berikut ke **node Leader** dan tampilkan ke layar
    `request_log()`
13. Jangan lupa untuk menggunakan command berikut untuk revert konfigurasi `qdisc` pada `tc`

```
tc qdisc del dev lo root netem delay 1000ms 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://doc.rust-lang.org/book/ - Rust book (dasar-dasar Rust)
6. https://youtu.be/c_5Jy_AVDaM - Rust Axum tutorial (kind of?)