Tanmay Pramanick – A20541164

PA2 – Peer-to-peer publisher subscriber system

<u>DESIGN DOCUMENT</u>

1. Introduction

This document outlines the design of the Peer-to-Peer (P2P) Topic Publisher-Subscriber System, developed as part of the Advanced Operating Systems course. This system builds upon previous server-client architecture to create a distributed publisher-subscriber network with peer nodes and an indexing server. The system ensures efficient message sharing across peers, scalability, and fault tolerance, while maintaining low latency and high throughput.

2. Architectural Overview

The system consists of the following components:

- <u>Peer Nodes</u>: Each peer node is capable of hosting topics, publishing messages, and allowing clients to subscribe to topics. Peer nodes communicate directly with one another after an initial discovery phase using the indexing server.
- <u>Indexing Server</u>: The indexing server acts as a central registry, keeping track of active peer nodes and their hosted topics. It facilitates topic discovery by directing peers to the node responsible for a specific topic.
- <u>Clients</u>: Clients can act as publishers or subscribers, interacting with peer nodes to publish messages or pull messages from a subscribed topic.

3. Component Description and Responsibilities

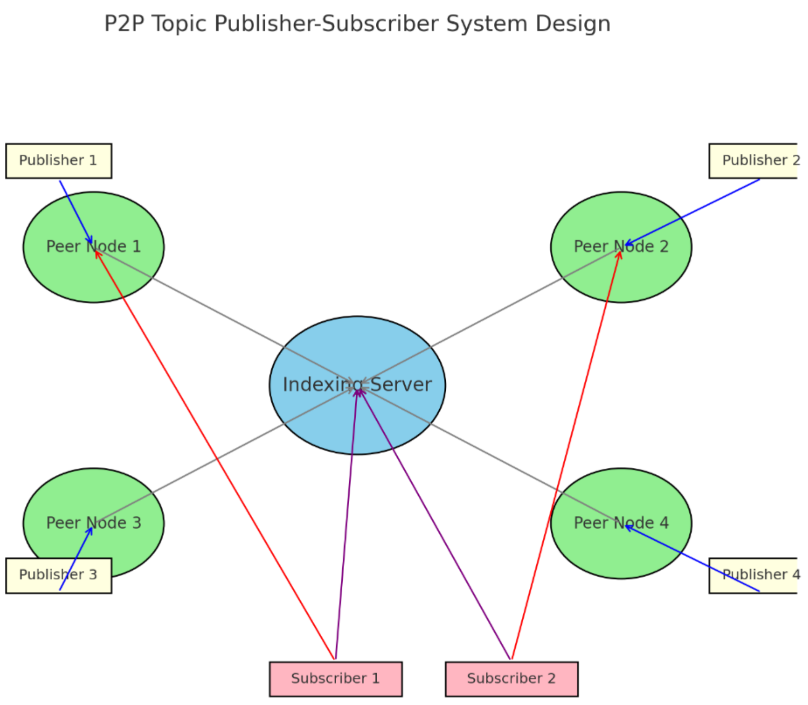
- Peer Node:
    - Listens for incoming connections using a designated IP address and port.
    - Hosts topics and registers them with the indexing server.
    - Allows publishers to publish messages to hosted topics.
    - Responds to queries from other peers or subscribers regarding available topics.
    - Implements graceful shutdown, unregistering hosted topics from the indexing server and deletes the topic from the Indexing Server.
- Indexing Server:
    - Maintains a list of active peer nodes and the topics they host.
    - Supports registration and shutting down of peer nodes.
    - Handles topic updates (e.g., new topics created, or existing topics deleted).
    - Responds to topic finding queries from peer nodes or clients.

- Client (Publisher/Subscriber):
    - Registers with the peer node and can create topics.
    - Publishes messages to topics hosted on peer nodes.
    - Subscribes to topics and pulls messages from peer nodes hosting those topics.

4. Design Decisions and Trade-offs

- Central Indexing Server vs. Decentralized Findings: We opted for a central indexing server to simplify topic findings. While a fully decentralized P2P system would eliminate the single point of failure, it would increase the complexity of peer findings and data consistency. The indexing server offers a balance between simplicity and scalability.
- Topic Storage Policy: Topics are stored on the peer that creates them. This minimizes the overhead of moving topics across nodes but may cause uneven distribution of topics among peers. Load balancing is a potential improvement for future versions.
- Concurrency and Parallelism: Peer nodes are implemented using asyncio to handle multiple requests concurrently. This design decision allows the peer nodes to manage multiple publishers and subscribers efficiently without resorting to more resource-intensive threading.

5. System Flow



1. Indexing Server Start-Up: The indexing server is started first to maintain a registry of all peers and their topics.
2. Peer Nodes Registration: Peer nodes start and register themselves with the indexing server, providing their IP and port along with the list of topics they host.

3. Client Interactions:

Publishers -

Publishers create topics on specific peer nodes and create topics and publish messages to that topic, also delete some topics.

Subscribers -

Subscribers query the indexing server for the desired topic, get the peer node information from the indexing server, and connect to that peer to subscribe and pull messages.

6. APIs and Functionality

- PeerNode API:
  - Register Topic: Registers a topic with the indexing server.
  - Publish Message: Publishes a message to a specific topic hosted by the peer.
  - Subscribe: Allows clients to subscribe to a hosted topic.
  - Pull Messages: Retrieves all messages for a subscribed topic.
- Indexing Server API:
  - Register Peer: Registers a new peer node and the topics it hosts.
  - Unregister Peer: Removes a peer node and its topics from the registry.
  - Query Topic: Provides information about which peer hosts a given topic.
- Client API:
  - Create Topic: Sends a request to create a new topic on the peer node.
  - Delete Topic: Sends a request to delete a topic hosted by the peer node.
  - Send Message: Publishes a message to the specified topic.
  - Subscribe: Subscribes to a topic to receive future messages.
  - Pull messages: Retrieves messages from the subscribed topic.

- Sufficiency of APIs:

The provided APIs effectively allow for full functionality in a P2P publisher-subscriber model. The APIs cover topic registration, publication, subscription, and message retrieval. However, future extensions could include:

- Load Balancing: Adding APIs for rebalancing topics across multiple peer nodes when one shuts down or when the system scales could further improve resilience.
- Peer Failure Handling: APIs for peer health checks or topic migration during a peer failure could be helpful in improving availability and reliability.

7. Bottlenecks and Improvements

Identified Bottlenecks

- Centralized Indexing Server: A primary bottleneck is the centralized indexing server. This server could become a performance bottleneck as the number of peers and topics scales. During testing, it was observed that the latency for topic queries increased as the number of concurrent requests grew.
- Network Latency: As the peers communicate over a network, network latency inherently affects message publication and subscription efficiency, particularly when peers are deployed across distributed environments.
- Peer Node Load: Each peer node can only handle a limited number of clients concurrently. During peak times, peers may become overloaded, leading to higher response times.

Potential Improvements

- Distributed Hash Table (DHT): Instead of a centralized indexing server, a DHT could be used for peer discovery to improve scalability and eliminate the single point of failure.
- Replication: Topics could be replicated across multiple peer nodes to improve reliability in case of node failure. This would require additional APIs to synchronize messages across replicated nodes.
- Load Distribution: A load-balancing mechanism could be introduced to evenly distribute topics among peers, allowing the system to dynamically adjust to load changes and avoid any single peer becoming a bottleneck.
- Enhanced Peer-to-Peer Communication: Allowing direct peer-to-peer communication between nodes without always going through the indexing server for lookups could improve efficiency, particularly for frequently accessed topics.

9. Future Extensions

In the future, the system could be extended in the following ways:

- **Authentication and Authorization**: Adding security mechanisms to ensure that only authorized clients can publish or subscribe to topics.
- **Advanced Failure Recovery**: Implementing advanced failure recovery, such as topic migration and state synchronization, would help maintain system stability even when peers leave unexpectedly.

- **Performance Optimization**: Improving the concurrency model with advanced techniques such as thread pools or using a reactive programming framework to further reduce latency and enhance throughput.

10. Conclusion

The designed P2P Topic Publisher-Subscriber System provides an efficient way to distribute and manage messages across multiple peer nodes. The use of an indexing server simplifies the finding process, while the asyncio-based architecture ensures that nodes can handle multiple requests concurrently. The system achieves scalability while maintaining low latency and offers avenues for further improvements, including decentralization and better load distribution.

This system is functional and offers basic publish/subscribe capabilities with centralized topic management. While the current design achieves the assignment's goals, further work is needed to fully decentralize the architecture and address potential bottlenecks as outlined above