Assignment - Distributed Computing - SE5090

Name	Muthumala A.K
Reg Number	MS24906562

Introduction

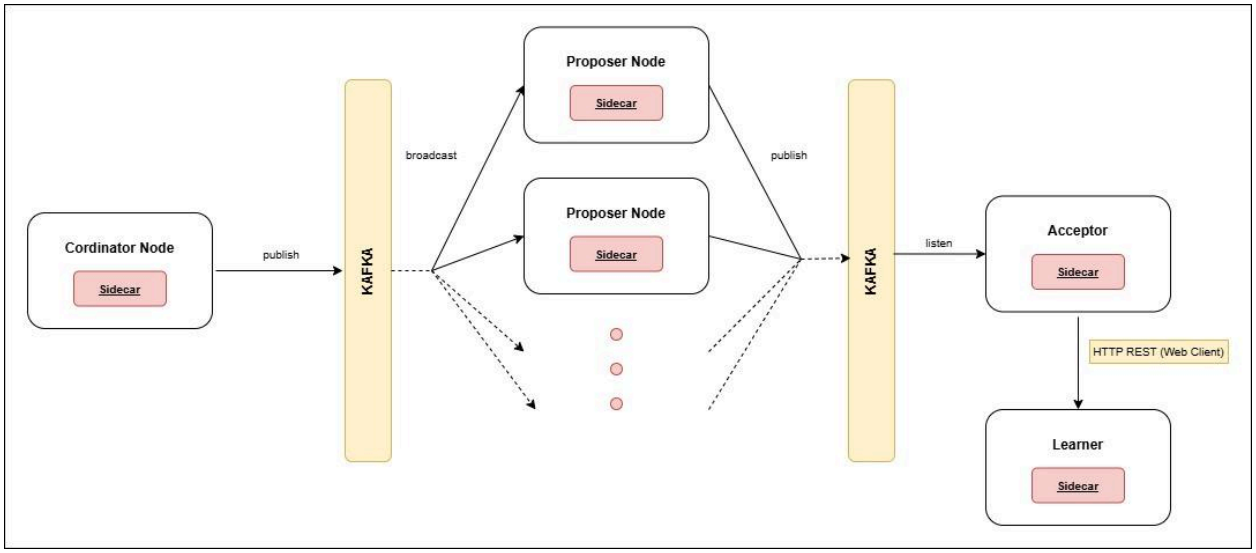
This report presents the design and implementation of a distributed word counting system developed for the SE5090 – Distributed Computing course. The system processes a document across multiple nodes, each responsible for counting words that begin with specific letter ranges. Roles in the system include Coordinator, Proposer, Acceptor, and Learner, inspired by distributed algorithms and consensus models.

The assignment demonstrates concepts such as scalability, fault tolerance, and inter-node communication using a sidecar proxy. Technologies like Spring Boot and Kafka are used to build and manage the cluster. The report outlines the solution architecture, communication flow, failure handling, and implementation details.

Outline of the Approach

- **Coordinator Node:** Detects active proposers, acceptors, and the learner. Assigns letter ranges and manages cluster metadata.
- **Proposer Nodes:** Receive lines from the coordinator and count words that start with letters within their assigned range.
- **Acceptor Nodes:** Validate word counts from proposers.
- **Learner Node:** Aggregates final validated counts and compiles the result.
- **Sidecar Proxy:** Each node has a sidecar handling logging and inter-node communication using REST over HTTP.

Solution Design



Technologies Used

- **Language & Framework :** Java (Spring Boot)
- **Messaging & Microservice Communication :** Kafka, WebClient (Spring Boot)
- **Serialization :** JSON
- **Logging :** Logback through sidecar
- **Communication Protocol :** REST, HTTP

Node Responsibilities

- **Coordinator :**
  Monitors node registration (via Kafka topics).
  Assigns character ranges dynamically based on node count.
  Broadcasts document lines to proposers.

- **Proposer :**
  Filters and counts words by range.
  Forwards results to acceptors.

- **Acceptor :**
  Validates counts and sends validated counts to the learner.

- **Learner:**
  Maintains global state.
  Displays final output grouped by starting letter.

Sidecar Implementation

In this solution, each microservice node (Coordinator, Proposer, Acceptor, Learner) includes a built-in sidecar component, implemented as a separate class within the service's codebase. Rather than being a standalone process, the sidecar is tightly integrated into each microservice and encapsulated in its own package for modularity.

Responsibilities of the Sidecar

- **Messaging Integration :** Handles communication between nodes via Apache Kafka, subscribing and publishing to relevant topics on behalf of the node.

- **Encapsulation of Communication Logic :** Abstracts the Kafka producer and consumer setup, serialization/deserialization, and topic routing from the main business logic.

- **Structured Logging :** Logs all incoming and outgoing messages with timestamps, message types, node IDs, and content summaries for debugging and analysis.

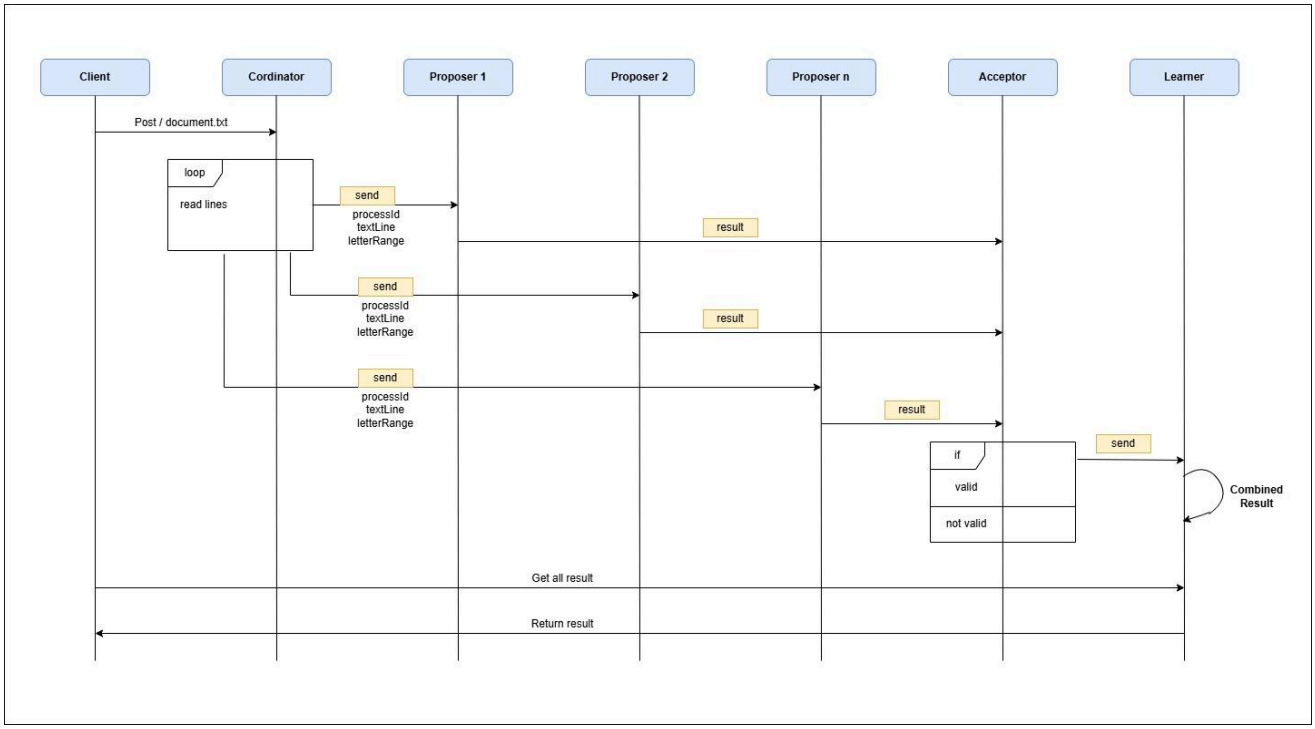
- **Modularity :** Ensures that communication and logging concerns are separated from core logic, making the service easier to maintain and test.

Algorithm for Letter Range Assignment (For Proposer Nodes)

Var letters = Array[a-z]
Var n = number of active proposer nodes
Var partitionSize = ceil(letters.length() / n)

Assign Ranges :
Node 1 : a to (a + partitionSize - 1)
Node 2 : next range
……

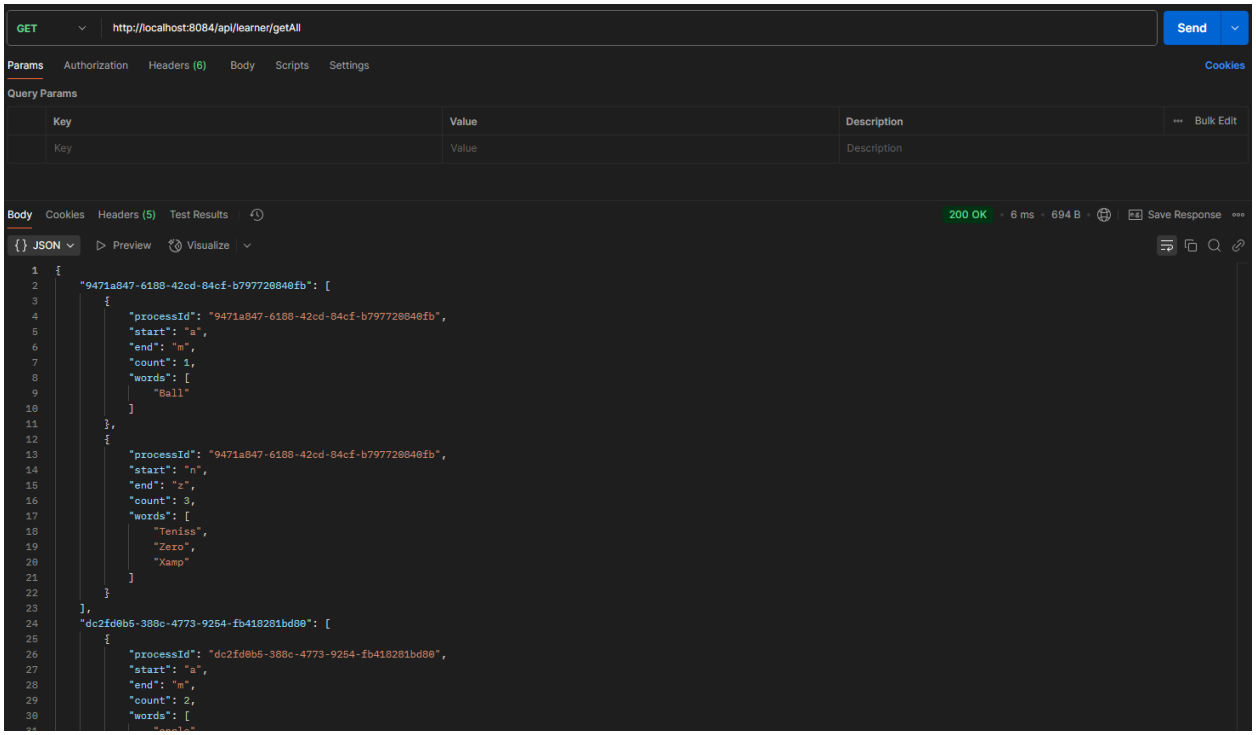
Sequence Diagram



Failure Scenarios and Fault Tolerance

Scenario	Fault Tolerance Strategy
Proposer Failure	If a proposer node crashes during processing, the Coordinator detects the failure and dynamically reassigns the corresponding letter range to another active proposer node. The document line is not retried. Instead, the new node will continue from the current point onward.
Acceptor Failure	Kafka's publish-subscribe mechanism ensures that if an acceptor is unavailable, others subscribed to the topic can still validate the proposer's result. No retries are triggered.
Learner Failure	Learner node state can be replicated or stored in persistent storage. If it crashes, a backup or new learner can resume processing from the last known state.
Network Failure	No retry is performed. Kafka handles message buffering and delivery. Nodes reconnect and resume processing as they become available.

Results



GitHub Link

Link : https://github.com/Asitha-Muthumala/Distributed-Word-Counter

References

- https://kafka.apache.org/
- https://medium.com/ms-club-of-sliit/lets-build-a-microservice-with-spring-boot-faf39b968857