Distributed Architecture Projects
Project 4.
Epidemic replication

Adrian Osedowski

Jerzy Jastrzebiec-Jankowski

February 22, 2023

1 Introduction

In this system, nodes belonging to the core layer have the most modern versions of the object. Instead, nodes belonging to the second layer have the eldest versions. Every node has their own file where all versions are stored. We built a system that contains 3 nodes of a core layer: A1, A2, A3. Four nodes are in backup layer. Backup layer consists of two sub-layers - Layer 1: B1, B2 and Layer 2: C1, C2.

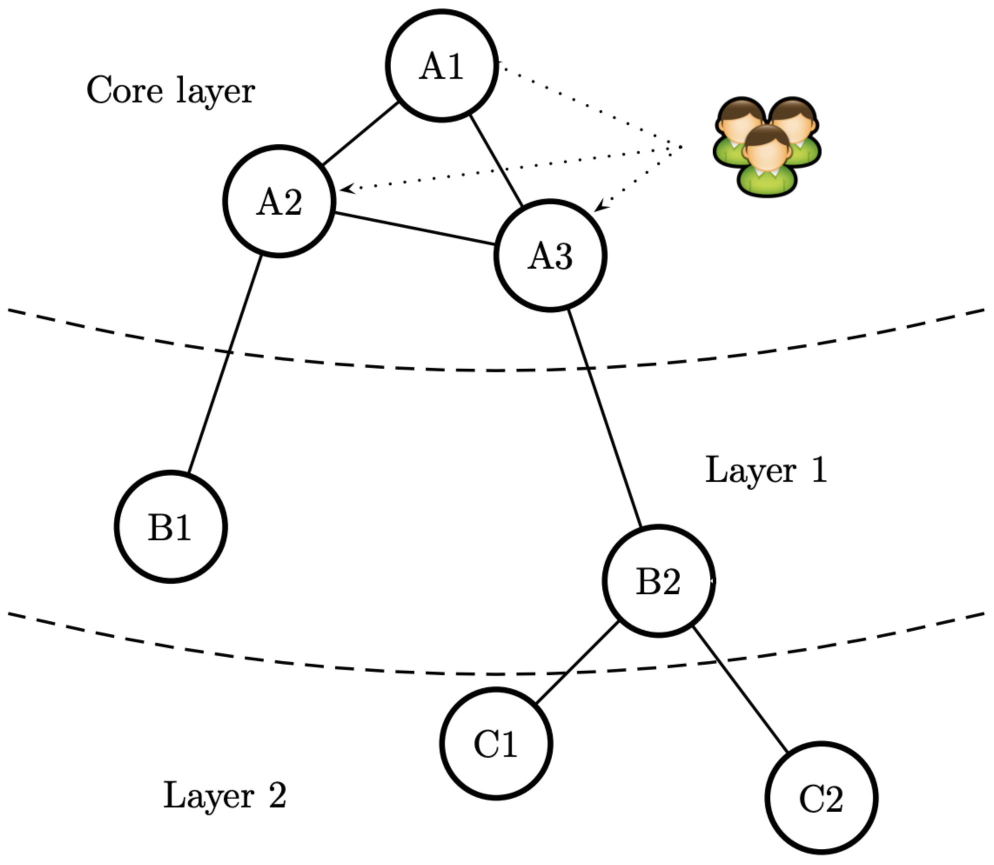

Figure 1: Connection diagram

2 Core Layer

We have three nodes in the core layer and they are exchanging token every 1 second, apart from the moments when the socket is blocked, what means that customer is connected and is doing write or read operations. After it finishes saving changes into log file as a new version the token is sent to next core node.
To be sure, that every node has the same data, we send last two versions from file with token. Then node which receives them, checks if it already has this version in log file. Then it adds one, two last versions or nothing. Thanks to that we are sure that we have the same versions every time in every node.

3 Backup

Layer 1 and Layer 2 are listening all the time for backup and then update theirs files, deleting old one and writing lines only with data received. Core Node sends backup to Layer 1 every ten updates, which means every ten times the customer connects with them, also means every 10 new lines written to file of cores. Layer 1 sends backup every 10 seconds to Layer 2. Layer 1 and 2 also listen all the time for connections from Customer, but they can process read-only requests.

4 Customer

Customer takes lines written in the .txt file, and then depending on the passed argument - connects to the specified node and processes passed operations.
This transactions can be of two types:

- read-only: $b < f >; r(30); r(49); r(69); c$
  b means begin, c close, r read with argument defining the line from which we want to read, ¡f¿ is an integer that contains the value 0, 1 o 2 and indicates the layer where the transaction will run, 0 means
- No read-only: $b; r(12); w(49, 53); r(69); c$
  These transactions will always run on one of the nodes of the core layer. w means write operation, where first argument is an line and second value

5 Classes

- CoreNode - functions of connection between core nodes, listening for customer, and main how to run an algorithm,
- SecondaryNode - funtions to send a backup and listen for it,
- LayerOne - main of B1 and B2, how they work,
- LayerTwo - main of C1 and C2, how they wor,
- Customer - processing a file with tranaction and connection with nodes,

6 WebSockets

WebSockets were used in the project to monitor the state of the nodes in real time. To do this we needed to introduce the WebSocket server and client. Client was a part of the main program and server was created in another standalone project.

6.1 Client

To send the messages from nodes we needed to create the WebSocket client. It was implemented as a class in the main project. In the class we have used the jakarta websocket package. The client was initialized by passing the address of the server and the id of the node. Each node was sending the messages to the server on their own endpoint so that the server can recognize which node is sending the message. To send the message and provide real time monitoring we have introduced the sendStatus method which was accepting the message string as an input and then was sending it to the server. The sendstatus method was executed every time the node was updating its value.

6.2 Server

To create the WebSocket server we have used the Spring application framework as it provides the WebSocket server functionalities in a very developer-friendly way. To configure the server we needed to implement the WebSocketConfigurer interface and override registerwebsockethandler method when we have defined the ULRs on which the server should listen. To know which node is sending the message we needed to extract the information from the endpoints. To do this we had to overwrite the beforeHandshake method. We had to also define the endpoint on which we are listening to the messages. To do this we have created the NodeEndpoint class which implements the TextWebSocketHandler.

7 Run programs

Fist we have to run WebSocketServer from different project. Next we have to run 2 programs LayerTwo with arguments 1 and 2, then 2 LayerOne with arguments 1 and 2. At the end we have to run two CoreNode with arguments 2 and 3. Last program must be CoreNode with argument equal to 1. Our system is running now, so we can run Customer as many times as we want. We need to modify Customer.txt file for processing different operations.