University of St.Gallen

Institute of Computer Science

Distributed Systems, Fall Semester 2023
Sanjiv Jha, Jérémy Lemée & Andrei Ciortea
{sanjiv.jha, jeremy.lemee, andrei.ciortea}@unisg.ch

Assignment 4: Vector Clocks (7pt)

Deadline: Nov 28, 2023; 23:59 CET

In this assignment, you will create a UDP-based (User Datagram Protocol) chat application that uses logical clocks to maintain the causal and temporal ordering of messages. To this end, you will implement two of the most well-known algorithms in distributed systems for consistent event ordering: Lamport's timestamp and the vector clocks algorithm.

To support this assignment, we provide you with a server-client implementation[1] that you will run on your local machine through the given command in the `README`. Once started, the server keeps on listening on **localhost** and default port **4040**.

(1) **A Chat Application using Lamport Timestamp (2pt)** Lamport Timestamps represent a simple algorithm to order distributed events partially (see Lecture 7). In this task, using the given project template, you will simulate a UDP client-server chat application[2] and implement Lamport Timestamps for seeing the partial ordering of the messages shared from the server to multiple clients. In this task, we will use four processes — three clients and one server. Once you update and build the project template, the server should send out capitalized forms of the messages sent by the clients with random delays. All of the messages received from the server have Lamport timestamps appended. Because the messages include local clocks, this implementation can help users extract a partial ordering of the events (send and receive) or messages.

Your task is to update the project template of the client's program `UdpLTClient.java` and `LTClientThread.java` to maintain the Lamport timestamp for the client clocks whenever they receive or send messages to the server. The clock should only be updated when the received clock value is greater than the clock value of the client. For every message sent or received (considering that it changes the state) by the client, increment its local clock by 1 i.e. **tick**. The Lamport timestamp for this task is implemented in `LamportTimestamp.java`[3]. Update the template to maintain the clocks and test it using the given test file `LamportTimestampTest.java` and the command from the `README`. An example of communication between the server and clients using Lamport timestamp is shown in Figure 1.

(2) **Message ordering using vector clocks (4pt)** For this task, your chat application will implement the same basic functionality as in *Task 1* — but using vector clocks to determine the ordering of the messages more precisely. The clock update rule remains the same in this implementation as well. However, for vector clock implementation `VectorClock.java`, all the processes maintain a vector (an `Array` of size = the number of participants) that logs the clocks of all clients involved in the chat. This vector form of the timestamps can help users deduce more precise ordering as well as a causal relationship between the consecutive messages.

Only the server maintains clocks for all four processes in this client-server chat application. The remaining three processes (clients) will only maintain the server and their local clocks. Thus, the clients should request the server to share the history of the group chat by sending a -**history**—

[1] The project template https://github.com/HSG-DS-HS23/Assignment4 is tested and implemented using Java 17 https://www.oracle.com/java/technologies/downloads/#java17

[2] Datagram sockets and functions in Java https://docs.oracle.com/javase/8/docs/api/java/net/DatagramSocket.html

[3] Logical clocks update functions https://aeroncookbook.com/distributed-systems-basics/logical-clocks/

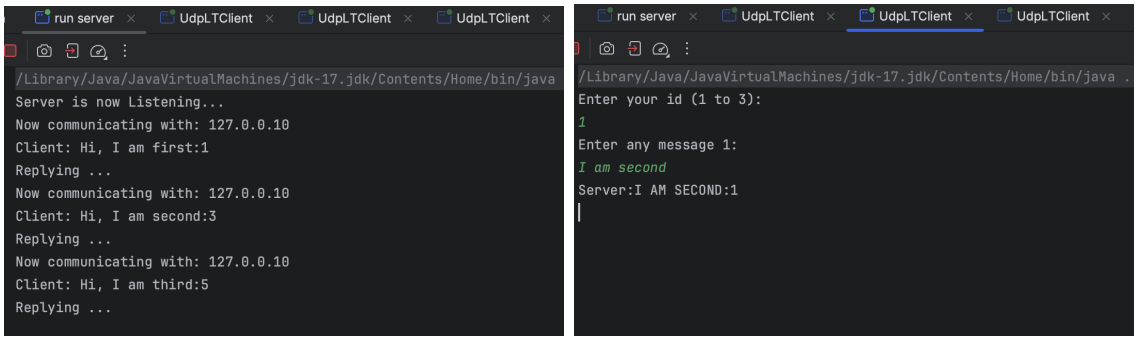

Figure 1: UDP client server using Lamport timestamp

message to understand the complete conversation (happened before joining or not shared with them). Remember that the UDP message server here is not sending a continuous message. Therefore, in this task, you would use the vector clocks of the given chat history to sort the events chronologically [4].

To do this, update the project template `UdpVectorClient.java` and `VectorClientThread.java`. You can test your updated Vector Clock implementation (`VectorClock.java`) using the provided test file `VectorClockTest.java` and the command on the `README`. Using the given project template, simulate a client-server chat application and display a sorted chat history for the users. An example of messaging among four processes and the sorted chat history (logs) can be seen in Figure 2. As a result of this task, provide a screenshot of any of the clients' output as `history-task2.png` that shows the sorted history and a portion of the communication (random chats, does not have to be the same as in Figure 2)) with the server.

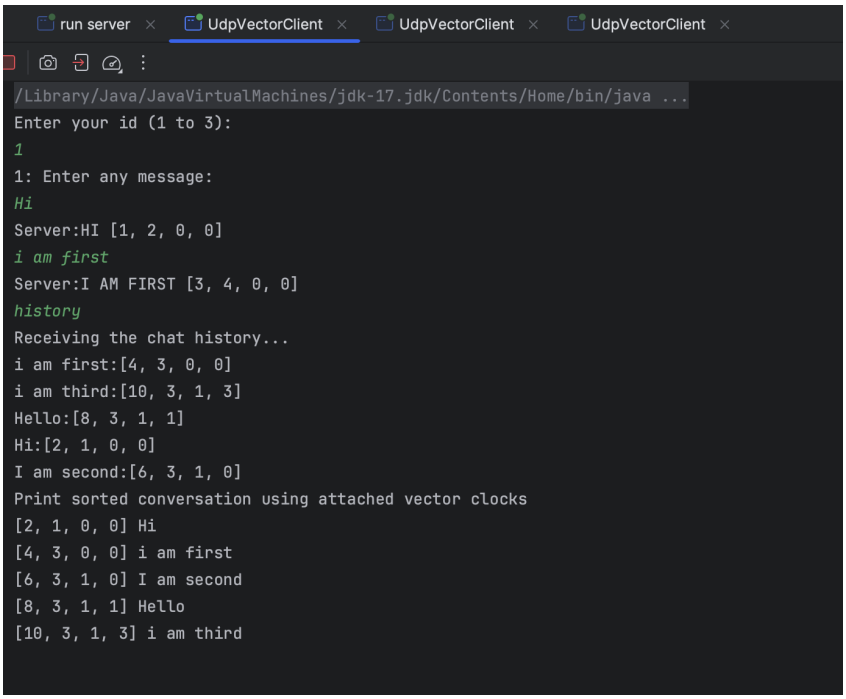

Figure 2: Client-server chat application using vector clock for sorting message history

③ Conceptual Evaluation(1pt)

1.) (0.25pt) What is causal consistency? Explain it using the happened-before relation.

[4] You could use custom comparators for sorting, examples https://www.delftstack.com/howto/java/sort-2d-array-java/, https://docs.oracle.com/javase/8/docs/api/java/util/List.html

2.) (0.25pt) You are responsible for designing a distributed system that maintains a partial ordering of operations on a data store (for instance, maintaining a time-series log database receiving entries from multiple independent processes/sensors with minimum or no concurrency). When would you choose Lamport timestamps over vector clocks? Explain your argument. What are the design objectives you can meet with both?

3.) (0.5pt) Vector clocks are an extension of the Lamport timestamp algorithm. However, scaling a vector clock to handle multiple processes can be challenging. Propose some solutions to this and explain your argument. To help you dive deeper into the topic, you can have a quick look at some of the issues raised in the following paper by Landes Tobias. "Dynamic Vector Clocks for Consistent Ordering of Events in Dynamic Distributed Applications." https://vs.inf.ethz.ch/edu/HS2016/VS/exercises/A3/DVC_Landes.pdf.

Hand-in Instructions By the deadline, you should hand in a single **zip** file via Canvas upload. The name of this file should start with `a4` and contain the last names of all team members separated by underscores (e.g., `a4_jha_lemee_ciortea.zip`). It should **only** contain the following files:

- All answers to the assignment questions (**Task 3**) in the given `REPORT.md`; if you wish to submit your solution code via GitHub, please include a link to your GitHub repository as well.
- Well commented and updated project template. When built, all tests must pass, and your implementation must be able to sort the history (logs).
- for **Task 2**, a screenshot of the sorted history `history-task2.png`.

Across all tasks in this and the other assignments in this course, you are **required to declare** any support that you received from others and, within reasonable bounds,[5] any support tools that you were using while solving the assignment.

[5]It is not required that you declare that you were using a text-editing software with orthographic correction; it is however required to declare if you were using any non-standard tools such as generative machine learning models such as GPT