Ira A. Fulton Schools of Engineering
Arizona State University

CSE 531: Distributed and Multiprocessor Operating Systems

Logical Clock Written Report

Name: Varad Vijay Deshmukh **ASU ID: 1225369184**

Problem Statement

This project focuses on the distributed system and gRPC implementation, like Project 1, with the primary goal of implementing logical clocks, which are local clocks for each process. This implementation is based on the illustrative diagram provided in Project 2 documentation. The project also involves integrating Lamport's Algorithm to ensure proper coordination between the logical clocks of customers and branches. In summary, the tasks for this project are as follows:

1. Implement logical clocks for all branch processes and customer processes.
2. Integrate Lamport's Logical Clock Algorithm to ensure coordination across all program processes.

Goal

The main goal of this project is to comprehensively grasp and apply the principles of Coordination. The utilization of logical clocks and the effective synchronization between them ensures seamless operation and proper coordination of the system. In this project, we acknowledge that the processes are not strictly sequential; instead, they are concurrent. Therefore, the use of logical clocks is crucial for monitoring requests and their processing, thereby safeguarding the system from potential failures.

Setup

1. Setup Ubuntu on Windows with WSL 2
    - **Ubuntu 22.04.2**

2. Install Python
    - **Python 3.10.6**

3. Install Python libraries:
    - **grpcio==1.59.0**

- **grpcio-tools==1.59.0**
- **protobuf==4.24.3**

4. Download and Unzip the submitted zip file.
    - **CSE531_Project_2_Varad.zip**

5. Get into the folder
    - **cd CSE_531_Project_2_Varad**

6. Optional – Build proto file
    - **python3 -m grpc_tools.protoc -I. --python_out=. --grpc_python_out=. Relay.proto**

7. Run main.py file with input_10.json
    - **python3 main.py input_10.json**

8. After execution is complete the following output files will be created in the folder
    - **branch_output.json**
    - **customer_output.json**
    - **events_output.json**

Implementation Processes

The program contains following files:

1. Relay.proto
   Relay.proto is a proto which contains two rpc services, Cast and Deliver. Cast is branch-to-branch communication and Deliver is customer-to-branch communication. Once the proto file is built, two files are created Relay_pb2 and Relay_pb2_grpc. These files help in using grpc functions like Relay_pb2_grpc.relayServicer and Relay_pb2_grpc.relayStub.

2. Customer.py
   This file contains a class of same name, Customer, and following are the functions in the class:
    a. createStub:
        - The "createStub" function facilitates the creation of stub using grpc functions for customer.
    b. executeEvents:
        - The "executeEvents" function serves the purpose of establishing communication between customers and branches. It utilizes the stub to send each customer event via the "Deliver" function defined in the

branch. Additionally, it updates the local logical clock for each event transmitted.

c. output_customer:
   - The "output_customer" function is responsible for generating a list that contains all customer events and their corresponding logical clock timestamps. This is achieved by storing the events and their execution times in a list.

d. output_events_only
   - Similarly, the "output_events_only" function, like "output_customer," returns a list that provides a comprehensive record of events. This record includes attributes such as id, type, customer-request-id, and logical clock information.

3. Branch.py

   This file contains a class of same name, Branch, and following are the functions in the class:

   a. createStubs
      - The "createStubs" function's primary purpose is to compile a list of stubs for all other branches within each branch. These stubs play a pivotal role in establishing branch-to-branch communication since each stub is associated with a specific branch across all processes.

   b. Deliver
      - "Deliver" is an RPC service defined as a function in the Branch class. Its main objective is to facilitate customer-to-branch communication. The "Deliver" function invokes the "Update" function within itself, which, in turn, returns a response. This response is then sent to the customer to complete the RPC communication.

   c. Cast
      - "Cast" is an RPC service defined as a function within the Branch class. Its goal is to enable branch-to-branch communication. The "Cast" function is invoked within the "Propagate_Deposit" and "Propagate_Withdraw" functions. Depending on the interface calls, either Withdraw or Deposit to update the balance of the branch in which it is currently located.

   d. Branch_message_handler
      - The "Branch_message_handler" function plays a crucial role in determining whether to call "Propagate_Deposit" or "Propagate_Withdraw." This function is called within the "Deliver" function and is essential for propagation once the branch balance has been updated.

   e. Update
      - The "Update" function is used to decide which function to call among the "Query," "Withdraw," and "Deposit" functions based on the request's interface.

   f. Query, Withdraw, Deposit

- Each of these functions serves the purpose of their respective interfaces. "Query" provides the current branch balance, "Deposit" adds money to the balance based on the request, and "Withdraw" subtracts money from the balance based on the request. Additionally, all these functions update the logical clock by considering the maximum value between the branch's logical clock and the request's logical clock, incrementing it by one time unit (Lamport’s Algorithm).

g. Propagate_Deposit, Propagate_Withdraw
   - "Propagate_Deposit" and "Propagate_Withdraw" both invoke the "Cast" function for all stubs in the stub list to send propagation requests to all other branches. They also use the "request_to_branch" function to append the list of events indicating which customer-request-IDs have been sent to which branches in the stub list.

h. Sent_to_Branch, Recv_from_Branch:
   - Both functions primarily serve the purpose of recording branch-to-branch communication. "Sent_to_Branch" records which customer-request-IDs have been sent to which branches, while "Recv_from_Branch" records which customer-request-IDs have been received from which branches. Additionally, both these functions update the logical clock by considering the maximum value between the branch's logical clock and the request's logical clock, incrementing it by one time unit (Lamport’s Algorithm).

i. output_branch:
   - The "output_branch" function generates a list of events specific to the branch it is associated with.

j. output_events_only:
   - Similarly, the "output_events_only" function, like "output_branch," returns a list that provides a comprehensive record of events. This record includes attributes such as id, type, customer-request-id, and logical clock information.

4. main.py
   a. Start_Process:
      - "Start_Process" is a crucial component responsible for initiating both Branch processes and Customer processes. It utilizes the multiprocessing library to launch all the necessary processes. Each newly created process then invokes either the "run_branch" or "run_customer" functions to execute their respective tasks.

   b. run_branch:
      - The "run_branch" function is responsible for executing the functions defined in the Branch class. It also initiates a server for the specific branch. After processing, it writes the output generated by the branch into output JSON files.

   c. run_customer:

- The "run_customer" function oversees invoking the functions defined in the Customer class. Once the execution is complete, it proceeds to write the output generated by the customer into output JSON files.

Results

```
varad@DESKTOP-F765JFB:~$ cd CSE531_Project_2_Varad
varad@DESKTOP-F765JFB:~/CSE531_Project_2_Varad$ python3 main.py input_10.json

NOTE: Wait till all processes are terminated.

Starting Branch Processes

Server started. Listening on port : 50001
Server started. Listening on port : 50004
Server started. Listening on port : 50005
Server started. Listening on port : 50003
Server started. Listening on port : 50007
Server started. Listening on port : 50006
Server started. Listening on port : 50002
Server started. Listening on port : 50008
Server started. Listening on port : 50009
Server started. Listening on port : 50010

Starting Customer Processes

Output File customer_output.son has been created inside the folder
Output File branch_output.son has been created inside the folder
Output File events_output.son has been created inside the folder

Terminating Processes
varad@DESKTOP-F765JFB:~/CSE531_Project_2_Varad$
```

The figure displays the terminal output produced when running the "main.py" script with the input data provided in "input_10.json." It also illustrates the creation of output files as part of the execution process.

branch_output.json

customer_output.json

events_output.json

```
varad@DESKTOP-F765JFB:~/CSE531_Project_2_Varad$ python3 checker_part_1.py customer_output.json
Customer ID: 1
  Event ID: 1, Logical Clock: 1 (OK)
  Event ID: 2, Logical Clock: 2 (OK)
Customer ID: 2
  Event ID: 3, Logical Clock: 1 (OK)
  Event ID: 4, Logical Clock: 2 (OK)
Customer ID: 3
  Event ID: 5, Logical Clock: 1 (OK)
  Event ID: 6, Logical Clock: 2 (OK)
Customer ID: 4
  Event ID: 7, Logical Clock: 1 (OK)
  Event ID: 8, Logical Clock: 2 (OK)
Customer ID: 5
  Event ID: 9, Logical Clock: 1 (OK)
  Event ID: 10, Logical Clock: 2 (OK)
Customer ID: 6
  Event ID: 11, Logical Clock: 1 (OK)
  Event ID: 12, Logical Clock: 2 (OK)
Customer ID: 7
  Event ID: 13, Logical Clock: 1 (OK)
  Event ID: 14, Logical Clock: 2 (OK)
Customer ID: 8
  Event ID: 15, Logical Clock: 1 (OK)
  Event ID: 16, Logical Clock: 2 (OK)
Customer ID: 9
  Event ID: 17, Logical Clock: 1 (OK)
  Event ID: 18, Logical Clock: 2 (OK)
Customer ID: 10
  Event ID: 19, Logical Clock: 1 (OK)
  Event ID: 20, Logical Clock: 2 (OK)

Summary: 20 out of 20 answers are correct.
varad@DESKTOP-F765JFB:~/CSE531_Project_2_Varad$
```

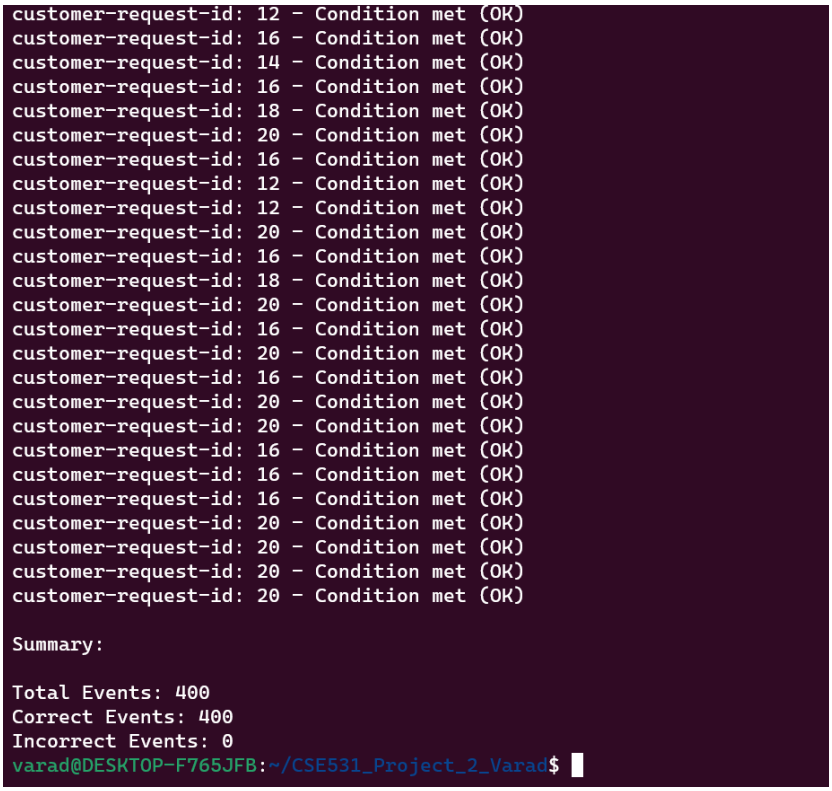
The figure presents the output of the "checker_part_1.py" script specifically for the "customer_output.json" file. It likely shows the results of validation of the data within the "customer_output.json" file using the script.

```
Branch ID: 10, Event ID: 19 - Conditions Met (OK)
Branch ID: 10, Event ID: 19 - Conditions Met (OK)
Branch ID: 10, Event ID: 15 - Conditions Met (OK)
Branch ID: 10, Event ID: 19 - Conditions Met (OK)
Branch ID: 10, Event ID: 19 - Conditions Met (OK)
Branch ID: 10, Event ID: 20 - Conditions Met (OK)
Branch ID: 10, Event ID: 10 - Conditions Met (OK)
Branch ID: 10, Event ID: 20 - Conditions Met (OK)
Branch ID: 10, Event ID: 6 - Conditions Met (OK)
Branch ID: 10, Event ID: 2 - Conditions Met (OK)
Branch ID: 10, Event ID: 20 - Conditions Met (OK)
Branch ID: 10, Event ID: 20 - Conditions Met (OK)
Branch ID: 10, Event ID: 14 - Conditions Met (OK)
Branch ID: 10, Event ID: 18 - Conditions Met (OK)
Branch ID: 10, Event ID: 20 - Conditions Met (OK)
Branch ID: 10, Event ID: 12 - Conditions Met (OK)
Branch ID: 10, Event ID: 20 - Conditions Met (OK)
Branch ID: 10, Event ID: 20 - Conditions Met (OK)
Branch ID: 10, Event ID: 16 - Conditions Met (OK)
Branch ID: 10, Event ID: 20 - Conditions Met (OK)
Branch ID: 10, Event ID: 20 - Conditions Met (OK)
Branch ID: 10, Event ID: 20 - Conditions Met (OK)

Summary:

Total Events: 380
Correct Events: 380
Incorrect Events: 0
varad@DESKTOP-F765JFB:~/CSE531_Project_2_Varad$
```

The figure presents the output of the "checker_part_2.py" script specifically for the "branch_output.json" file. It likely shows the results of validation of the data within the "branch_output.json" file using the script.



The figure presents the output of the "checker_part_3.py" script specifically for the "events_output.json" file. It likely shows the results of validation of the data within the "events_output.json" file using the script.