Ira A. Fulton Schools of Engineering
Arizona State University

CSE 531: Distributed and Multiprocessor Operating Systems

Logical Clock Written Report

Problem Statement

In distributed systems, achieving synchronization and maintaining a consistent order of events across different processes is a significant challenge. Without a proper mechanism to order these events, distributed systems can suffer from inconsistencies and errors, especially in the context of concurrent operations and interactions. The absence of a centralized clock in such systems further complicates the task of maintaining a chronological order of events. In this project, we use a banking system as an example to explore how to deal with this problem.

Goal

The primary objective is to address these challenges by implementing Lamport's logical clock algorithm in a system consisting of customer and branch processes. The specific goals are:

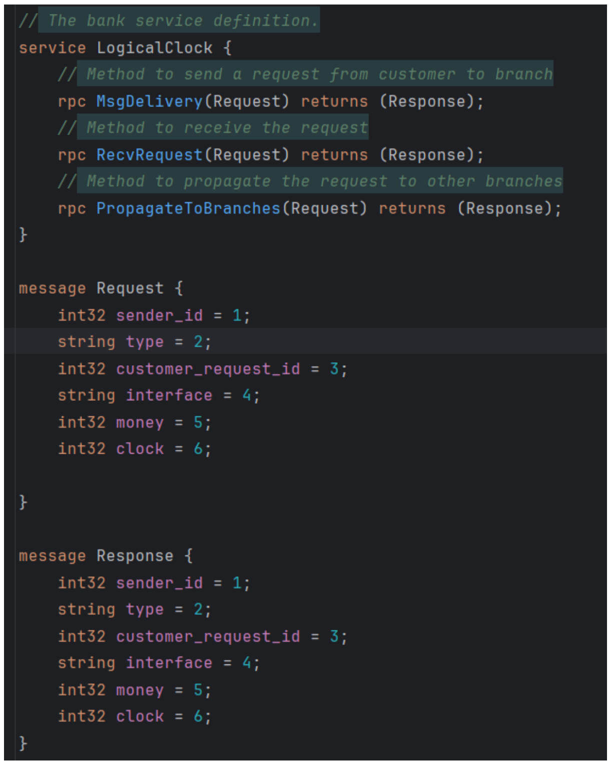
- **Implement Logical Clocks**: Integrate logical clocks into every customer and branch process. Logical clocks are a mechanism to order events in a distributed system without relying on physical time.
- **Lamport's Algorithm for Clock Coordination**: Apply Lamport's algorithm to coordinate clocks among the processes. The algorithm provides a way to order events across different processes in a distributed system, ensuring that all processes have a consistent view of the order of events.

Setup

For this project, I use gRPC and Python 3.12. How to set up gRPC and Python environment has been covered in Project 1 report. In the same environment built in Project 1, I made the following modifications:

1. **Redefine Protocol Buffers (Proto) File:**

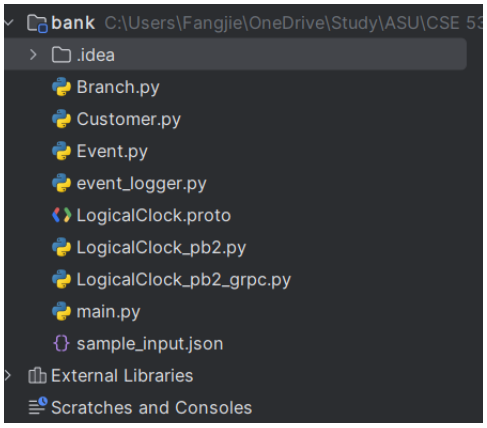
Create a **LogicalClock.proto** file that defines the methods interface and message types. I have defined "Request" and "Response" messages for customer – branch or branch-branch communications, and main methods as follows: MsgDelivery, RecvRequest and PropagateToBranches.



```
// The bank service definition.
service LogicalClock {
    // Method to send a request from customer to branch
    rpc MsgDelivery(Request) returns (Response);
    // Method to receive the request
    rpc RecvRequest(Request) returns (Response);
    // Method to propagate the request to other branches
    rpc PropagateToBranches(Request) returns (Response);
}

message Request {
    int32 sender_id = 1;
    string type = 2;
    int32 customer_request_id = 3;
    string interface = 4;
    int32 money = 5;
    int32 clock = 6;

}

message Response {
    int32 sender_id = 1;
    string type = 2;
    int32 customer_request_id = 3;
    string interface = 4;
    int32 money = 5;
    int32 clock = 6;
}
```

2. **Generate gRPC Code:**

Compile the **LogicalClock.proto** file to generate **LogicalClock_pb2.py** and **LogicalClock _pb2_grpc.py** by using the protocol compiler. Those two files include all the methods that can be used in class files and main process files.



```
bank C:\Users\Fangjie\OneDrive\Study\ASU\CSE 53
  .idea
  Branch.py
  Customer.py
  Event.py
  event_logger.py
  LogicalClock.proto
  LogicalClock_pb2.py
  LogicalClock_pb2_grpc.py
  main.py
  sample_input.json
External Libraries
Scratches and Consoles
```

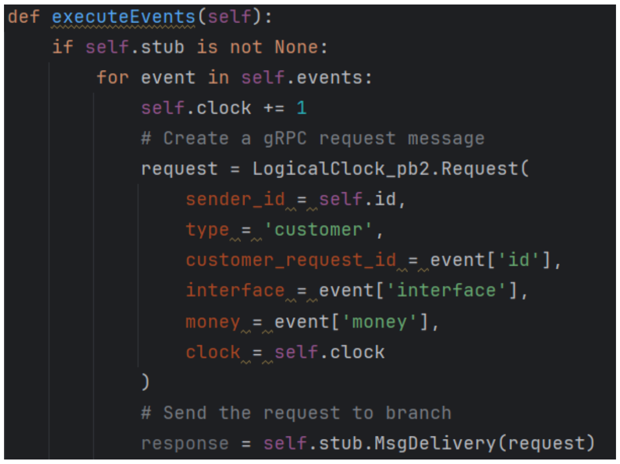
Implementation Processes

The implementation is structured into several Python files, each serving a specific purpose within the system.

1. Customer Class (Customer.py)

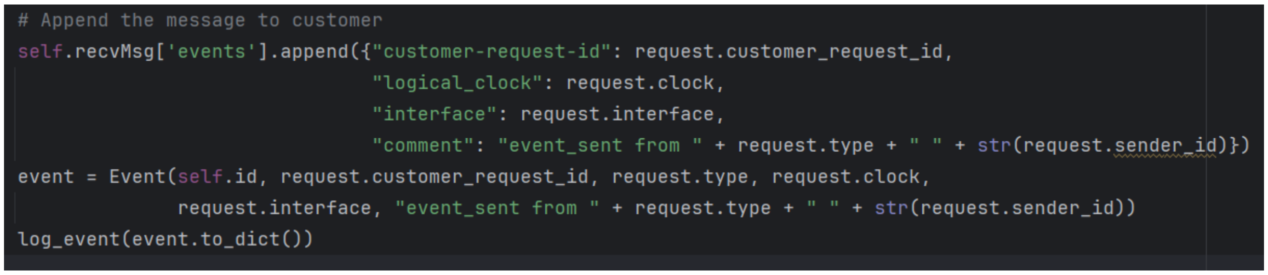
Represents a customer in the banking system. It is responsible for sending requests to branches and maintaining a logical clock for each event.

- **createStub Method**: Establishes a gRPC channel and stub for communication with the branch server.
- **executeEvents Method**:
  Iterates over customer events, increments the logical clock, and sends requests to the branch.



```python
def executeEvents(self):
    if self.stub is not None:
        for event in self.events:
            self.clock += 1
            # Create a gRPC request message
            request = LogicalClock_pb2.Request(
                sender_id = self.id,
                type = 'customer',
                customer_request_id = event['id'],
                interface = event['interface'],
                money = event['money'],
                clock = self.clock
            )
            # Send the request to branch
            response = self.stub.MsgDelivery(request)
```

Each event is logged using the log_event function from event_logger.

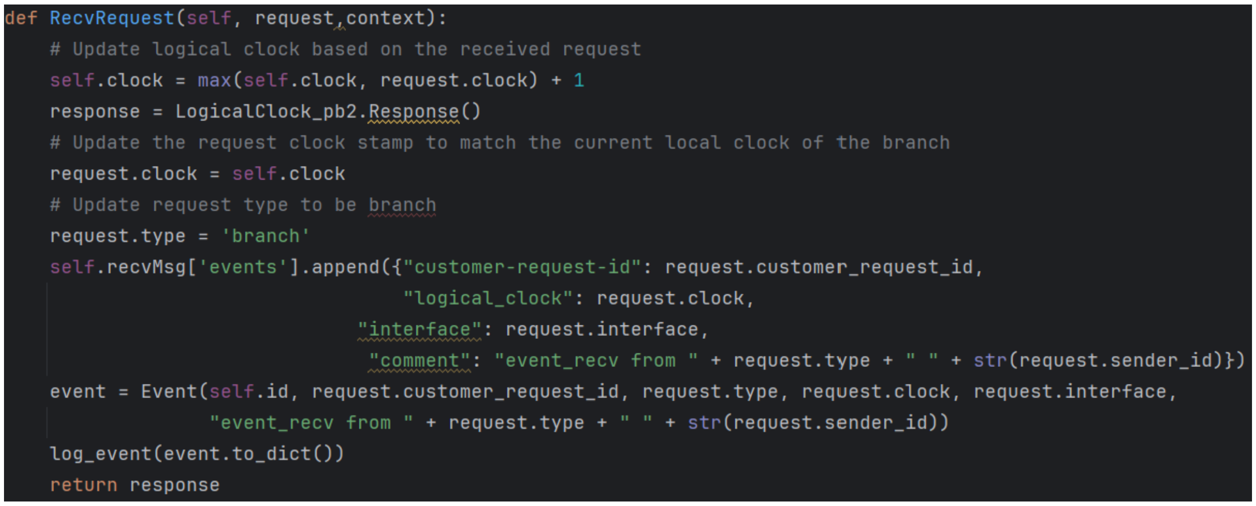


```python
# Append the message to customer
self.recvMsg['events'].append({"customer-request-id": request.customer_request_id,
                               "logical_clock": request.clock,
                               "interface": request.interface,
                               "comment": "event_sent from " + request.type + " " + str(request.sender_id)})
event = Event(self.id, request.customer_request_id, request.type, request.clock,
              request.interface, "event_sent from " + request.type + " " + str(request.sender_id))
log_event(event.to_dict())
```

2. Branch Class (Branch.py)

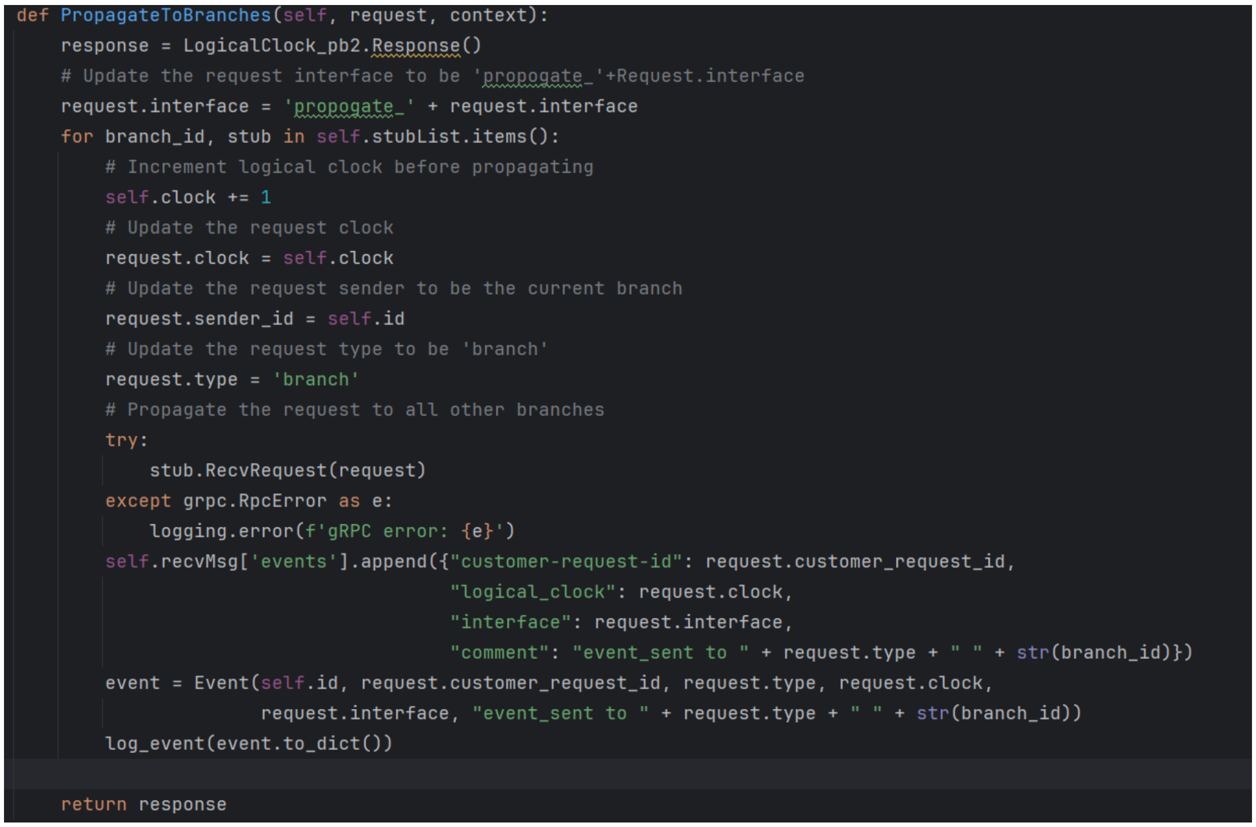
Represents a branch in the banking system. Each branch processes requests from customers and propagates these requests to other branches. It also maintains a logical clock for each event during those branch-related processes.

- **RecvRequest Method**: Processes incoming requests, updates the logical clock, and logs the event. This method is pivotal in receiving and responding to customer requests.



```python
def RecvRequest(self, request,context):
    # Update logical clock based on the received request
    self.clock = max(self.clock, request.clock) + 1
    response = LogicalClock_pb2.Response()
    # Update the request clock stamp to match the current local clock of the branch
    request.clock = self.clock
    # Update request type to be branch
    request.type = 'branch'
    self.recvMsg['events'].append({"customer-request-id": request.customer_request_id,
                                    "logical_clock": request.clock,
                                "interface": request.interface,
                                 "comment": "event_recv from " + request.type + " " + str(request.sender_id)})
    event = Event(self.id, request.customer_request_id, request.type, request.clock, request.interface,
                  "event_recv from " + request.type + " " + str(request.sender_id))
    log_event(event.to_dict())
    return response
```

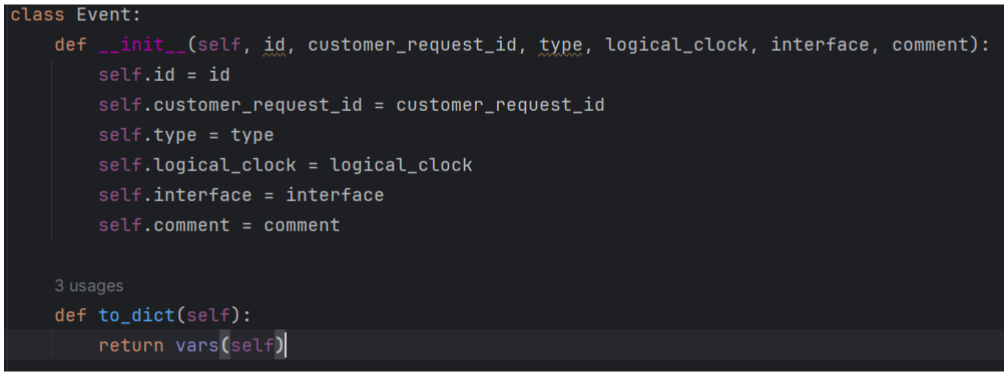
- **PropagateToBranches Method**: Propagates the request to other branches, ensuring synchronization across the system. Each propagation event is logged.



```python
def PropagateToBranches(self, request, context):
    response = LogicalClock_pb2.Response()
    # Update the request interface to be 'propogate_'+Request.interface
    request.interface = 'propogate_' + request.interface
    for branch_id, stub in self.stubList.items():
        # Increment logical clock before propagating
        self.clock += 1
        # Update the request clock
        request.clock = self.clock
        # Update the request sender to be the current branch
        request.sender_id = self.id
        # Update the request type to be 'branch'
        request.type = 'branch'
        # Propagate the request to all other branches
        try:
            stub.RecvRequest(request)
        except grpc.RpcError as e:
            logging.error(f'gRPC error: {e}')
        self.recvMsg['events'].append({"customer-request-id": request.customer_request_id,
                                       "logical_clock": request.clock,
                                       "interface": request.interface,
                                       "comment": "event_sent to " + request.type + " " + str(branch_id)})
        event = Event(self.id, request.customer_request_id, request.type, request.clock,
                      request.interface, "event_sent to " + request.type + " " + str(branch_id))
        log_event(event.to_dict())

    return response
```

3. Event Class (Event.py)

Defines the structure of an event, including its ID, customer request ID, type, logical clock, interface, and comment. This class facilitates consistent logging of events throughout the system.

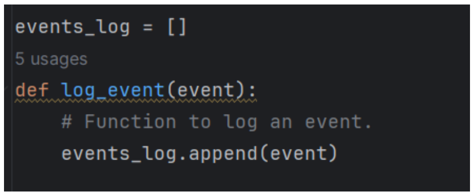


```
class Event:
    def __init__(self, id, customer_request_id, type, logical_clock, interface, comment):
        self.id = id
        self.customer_request_id = customer_request_id
        self.type = type
        self.logical_clock = logical_clock
        self.interface = interface
        self.comment = comment

    3 usages
    def to_dict(self):
        return vars(self)
```

4. Event Logger (event_logger.py)

This is a global file that stores all events logged during the execution of the system.

- **log_event Function**: Appends events to the events_log. This function is used across the Customer and Branch classes to log events consistently.



```
events_log = []
5 usages
def log_event(event):
    # Function to log an event.
    events_log.append(event)
```

5. Main Execution (main.py)

Reads the initial configuration from sample_input.json, initializes branch servers, and creates customer instances.

- **Execution**: Customers execute their events, which are processed by branches and propagated across the system.
- **Output**: It prints the logged events for each customer and branch, followed by a sorted list of all events based on customer request IDs and logical clocks.

Results

The results confirm the successful implementation of Lamport's logical clock algorithm in the distributed banking system. The system effectively manages the synchronization and ordering of events across multiple processes, ensuring consistency and reliability in handling customer requests and branch interactions.

1. **Part 1 - Customer Events:**

This section lists events initiated by customers, including the unique ID of the customer (id), the type of event (interface), and a comment describing the action (comment). The logical clock (logical_clock) increments with each event, indicating the ordering of events from the perspective of each customer.

The logical clock increment with each customer event illustrates the system's ability to maintain a sequential order of events from the customer's viewpoint.

Results:

```
{'id': 1, 'type': 'customer', 'events': [{'customer-request-id': 1, 'logical_clock': 1, 'interface': 'query',
'comment': 'event_sent from customer 1'}]}
```

```
{'id': 2, 'type': 'customer', 'events': [{'customer-request-id': 2, 'logical_clock': 1, 'interface':
'deposit', 'comment': 'event_sent from customer 2'}, {'customer-request-id': 3, 'logical_clock': 2,
'interface': 'query', 'comment': 'event_sent from customer 2'}]}
```

```
{'id': 3, 'type': 'customer', 'events': [{'customer-request-id': 4, 'logical_clock': 1, 'interface': 'query',
'comment': 'event_sent from customer 3'}]}
```

2. **Part 2 - Branch Events:**

This part shows the events processed by each branch. Events include receiving customer requests and propagating information to other branches. Each event is timestamped with the logical clock, demonstrating how branches synchronize their state with others.

The propagation of events and their logical timestamps validate the implementation of Lamport's algorithm in maintaining a consistent order of events across branches. The increasing logical clock values demonstrate the chronological processing and propagation of events.

Results:

```
{'id': 1,
'type': 'branch',

'events':

[{'customer-request-id': 1, 'logical_clock': 2, 'interface': 'query', 'comment': 'event_recv from branch 1'},

{'customer-request-id': 1, 'logical_clock': 3, 'interface': 'propogate_query', 'comment': 'event_sent to branch 2'},

{'customer-request-id': 1, 'logical_clock': 4, 'interface': 'propogate_query', 'comment': 'event_sent to branch 3'},

{'customer-request-id': 2, 'logical_clock': 7, 'interface': 'propogate_deposit', 'comment': 'event_recv from branch 2'},

{'customer-request-id': 3, 'logical_clock': 10, 'interface': 'propogate_query', 'comment': 'event_recv from branch 2'},

{'customer-request-id': 4, 'logical_clock': 14, 'interface': 'propogate_query', 'comment': 'event_recv from branch 3'}]}

{'id': 2,

'type': 'branch',

'events':

[{'customer-request-id': 1, 'logical_clock': 4, 'interface': 'propogate_query', 'comment': 'event_recv from branch 1'},

{'customer-request-id': 2, 'logical_clock': 5, 'interface': 'deposit', 'comment': 'event_recv from branch 2'},

{'customer-request-id': 2, 'logical_clock': 6, 'interface': 'propogate_deposit', 'comment': 'event_sent to branch 1'},

{'customer-request-id': 2, 'logical_clock': 7, 'interface': 'propogate_deposit', 'comment': 'event_sent to branch 3'},

{'customer-request-id': 3, 'logical_clock': 8, 'interface': 'query', 'comment': 'event_recv from branch 2'},
{'customer-request-id': 3, 'logical_clock': 9, 'interface': 'propogate_query', 'comment': 'event_sent to branch 1'},

{'customer-request-id': 3, 'logical_clock': 10, 'interface': 'propogate_query', 'comment': 'event_sent to branch 3'},

{'customer-request-id': 4, 'logical_clock': 15, 'interface': 'propogate_query', 'comment': 'event_recv from branch 3'}]}

{'id': 3,

'type': 'branch',

'events':

[{'customer-request-id': 1, 'logical_clock': 5, 'interface': 'propogate_query', 'comment': 'event_recv from branch 1'},

{'customer-request-id': 2, 'logical_clock': 8, 'interface': 'propogate_deposit', 'comment': 'event_recv from branch 2'},

{'customer-request-id': 3, 'logical_clock': 11, 'interface': 'propogate_query', 'comment': 'event_recv from branch 2'},

{'customer-request-id': 4, 'logical_clock': 12, 'interface': 'query', 'comment': 'event_recv from branch 3'},

{'customer-request-id': 4, 'logical_clock': 13, 'interface': 'propogate_query', 'comment': 'event_sent to branch 1'},

{'customer-request-id': 4, 'logical_clock': 14, 'interface': 'propogate_query', 'comment': 'event_sent to branch 2'}]}
```

3. Part 3 - Request Log:

This section presents a view of all events/requests sorted by customer_request_id and logical_clock. It provides a comprehensive timeline of how each request is handled across the system, from initial customer request to branch processing and inter-branch communications.

Sorting events by logical_clock helps us confirm that the system effectively synchronizes events across different processes. With the incremented logical_clock, the event processes happen in the logical order expected.

Results:

```
{'id': 1, 'customer_request_id': 1, 'type': 'customer', 'logical_clock': 1, 'interface': 'query',
'comment': 'event_sent from customer 1'}

{'id': 1, 'customer_request_id': 1, 'type': 'branch', 'logical_clock': 2, 'interface': 'query', 'comment':
'event_recv from branch 1'}

{'id': 1, 'customer_request_id': 1, 'type': 'branch', 'logical_clock': 3, 'interface': 'propogate_query',
'comment': 'event_sent to branch 2'}

{'id': 2, 'customer_request_id': 1, 'type': 'branch', 'logical_clock': 4, 'interface': 'propogate_query',
'comment': 'event_recv from branch 1'}

{'id': 1, 'customer_request_id': 1, 'type': 'branch', 'logical_clock': 4, 'interface': 'propogate_query',
'comment': 'event_sent to branch 3'}

{'id': 3, 'customer_request_id': 1, 'type': 'branch', 'logical_clock': 5, 'interface': 'propogate_query',
'comment': 'event_recv from branch 1'}

{'id': 2, 'customer_request_id': 2, 'type': 'customer', 'logical_clock': 1, 'interface': 'deposit',
'comment': 'event_sent from customer 2'}

{'id': 2, 'customer_request_id': 2, 'type': 'branch', 'logical_clock': 5, 'interface': 'deposit', 'comment':
'event_recv from branch 2'}

{'id': 2, 'customer_request_id': 2, 'type': 'branch', 'logical_clock': 6, 'interface': 'propogate_deposit',
'comment': 'event_sent to branch 1'}

{'id': 1, 'customer_request_id': 2, 'type': 'branch', 'logical_clock': 7, 'interface': 'propogate_deposit',
'comment': 'event_recv from branch 2'}

{'id': 2, 'customer_request_id': 2, 'type': 'branch', 'logical_clock': 7, 'interface': 'propogate_deposit',
'comment': 'event_sent to branch 3'}

{'id': 3, 'customer_request_id': 2, 'type': 'branch', 'logical_clock': 8, 'interface': 'propogate_deposit',
'comment': 'event_recv from branch 2'}
```

```
{'id': 2, 'customer_request_id': 3, 'type': 'customer', 'logical_clock': 2, 'interface': 'query',
'comment': 'event_sent from customer 2'}

{'id': 2, 'customer_request_id': 3, 'type': 'branch', 'logical_clock': 8, 'interface': 'query', 'comment':
'event_recv from branch 2'}

{'id': 2, 'customer_request_id': 3, 'type': 'branch', 'logical_clock': 9, 'interface': 'propogate_query',
'comment': 'event_sent to branch 1'}

{'id': 1, 'customer_request_id': 3, 'type': 'branch', 'logical_clock': 10, 'interface': 'propogate_query',
'comment': 'event_recv from branch 2'}

{'id': 2, 'customer_request_id': 3, 'type': 'branch', 'logical_clock': 10, 'interface': 'propogate_query',
'comment': 'event_sent to branch 3'}

{'id': 3, 'customer_request_id': 3, 'type': 'branch', 'logical_clock': 11, 'interface': 'propogate_query',
'comment': 'event_recv from branch 2'}

{'id': 3, 'customer_request_id': 4, 'type': 'customer', 'logical_clock': 1, 'interface': 'query',
'comment': 'event_sent from customer 3'}

{'id': 3, 'customer_request_id': 4, 'type': 'branch', 'logical_clock': 12, 'interface': 'query', 'comment':
'event_recv from branch 3'}

{'id': 3, 'customer_request_id': 4, 'type': 'branch', 'logical_clock': 13, 'interface': 'propogate_query',
'comment': 'event_sent to branch 1'}

{'id': 1, 'customer_request_id': 4, 'type': 'branch', 'logical_clock': 14, 'interface': 'propogate_query',
'comment': 'event_recv from branch 3'}

{'id': 3, 'customer_request_id': 4, 'type': 'branch', 'logical_clock': 14, 'interface': 'propogate_query',
'comment': 'event_sent to branch 2'}

{'id': 2, 'customer_request_id': 4, 'type': 'branch', 'logical_clock': 15, 'interface': 'propogate_query',
'comment': 'event_recv from branch 3'}
```