Winter22 CS271 Project2

January 24, 2022

Abstract

Global Snapshots are a way of capturing the entire state of a distributed application by recording the local state of each process along with the messages in each of the communication channels between processes. Global Snapshots are useful for creating checkpoints so we can restart from a consistent state in case of failures. Other uses for global snapshots include garbage collection, deadlock detection, and easier debugging.

In this project, you will use the Chandy-Lamport global snapshot algorithm to take global snapshots of a simple banking application that you will create. Each client will be able to send money, receive money and check their balance. **Additionally, each client will be capable of initiating a global snapshot using the Chandy-Lamport algorithm.**

1 Application Component

We will assume 4 clients. Each starts with a balance of $10.
A client can perform two banking operations:

- Transfer money from its account to another client. For example, if A transfers $x to B the following should occur
    1. A subtracts $x from its local balance (if A has insufficient balance it should print "INSUFFICIENT BALANCE")
    2. A sends a message to B with $x
    3. B increments its local balance by $
- Check its local balance

Your network connections should be represented by the directed graph in Figure 1. By looking at the figure we can see that A can send to B and receive messages from B, but that C can only send messages (not receive) to B. We can see that no messages can directly be exchanged between A and C. If we were

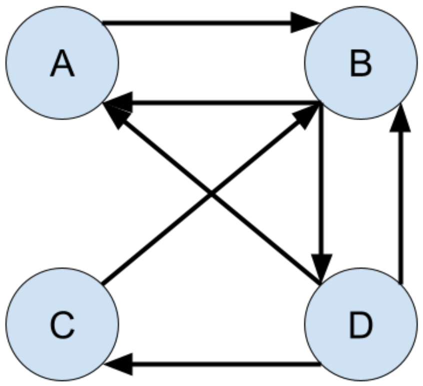

Figure 1: Network Connectivity Graph for 4 Clients. An arrow represents a unidirectional channel between two clients

to issue a transfer from A to C or B to C your program should print "NOT CONNECTED".

At any point during the execution, any client can take a global snapshot of the system using the Chandy-Lamport Algorithm. That client will be responsible for initiating the snapshot and then collecting and outputting the snapshot information to the user.

2 The Chandy-Lamport Algorithm

The Chandy-Lamport Algorithm is used to capture the local states of each process as well as the states of all of the channels between them. The resulting snapshot is guaranteed to be preserve happens-before relationships. In other words, if event e happens-before event f then for event f to be present in the snapshot, event e must also be present in the snapshot.

2.1 System Requirements

- The Chandy-Lamport Algorithm assumes lossless FIFO channels between all clients.
- Assume we have N clients that will not crash or fail.
- Taking a snapshot should not interfere with the normal application behavior (i.e. all clients should still be able to execute transactions during the snapshot).
- Any process can initiate a global snapshot.

What follows is a description of the Chandy-Lamport Algorithm.

2.2 Initiating a Snapshot

1. Process P_i wants to initiate a snapshot
2. P_i records its own local state
3. P_i sends a special MARKER messages on all outgoing channels
4. P_i starts recording incoming messages from incoming channels

2.3 Propagating a Snapshot

- If P_j receives a MARKER for the first time over channel C_{kj}
    1. P_j marks channel C_{kj} as empty
    2. P_j records its own local state
    3. P_j sends a special MARKER message on all outgoing channels
    4. P_j starts recording incoming messages from all incoming channels
- If P_j had already received a MARKER and receives another MARKER over channel C_{kj}
    1. P_j sets the state of the channel C_{kj} to contain all of the messages that were received on C_{kj} between when P_j received its first MARKER and when it received this MARKER.

2.4 Terminating a Snapshot

1. Once P_j has received MARKERS on all of its incoming channels it sends its snapshot consisting of its recorded local state and channel states to the initiating process P_i
2. Once P_i has received snapshots from all processes it concatenates them to form the global snapshot and prints it

Note: MARKER messages should be distinct from regular application messages. Since multiple clients may be issuing snapshots concurrently, MARKERS will need to carry a unique snapshot ID so MARKERS from different snapshots do not interfere with each other.

3 Implementation Detail Suggestions

1. Each client should keep track of its own balance.
2. Clients can assign globally unique IDs to each snapshot by concatenating their process ID and a local counter (e.g. *process_id.counter*). (See above why this is necessary)

4 User Interface

1. When starting a client, it should connect to all the other clients. You can provide a client's IP, or other identification info that can uniquely identify each client. Or this could be done via a configuration file or other methods that are appropriate.

2. Through the client user interface, we can issue transfer or balance transactions to an individual client. Once a client receives the transaction request from the user, the client executes it and displays on the screen "SUCCESS" or "INCORRECT" (for transfer transactions) or the balance (for balance transactions).

3. Through the client user interface, we can begin a snapshot. Once the snapshot has been taken you should print the local state of all of the clients (i.e. their respective balances) as well as any messages in any of the channels between them (e.g. transfer messages or MARKERs from other snapshots).

4. You should log all necessary information on the console for the sake of debugging and demonstration, e.g. Message sent to client XX. Message received from client YY. When a client issues a transaction, output its current balance before and after. When a MARKER is received print it out.

5. **You should a 3 seconds delay when sending a message.** This simulates the time for message passing and makes it easier for demoing concurrent events.

6. Use message passing primitives TCP/UDP. You can decide which alternative and explore the trade-offs. We will be interested in hearing your experience.

5 Demo Case

For the demo, you should have 4 clients. Initially, they should all have a balance of $10.
Here is an example demo case:

1. C initiates a snapshot

2. 1 second later A sends $5 to B

3. B receives the marker from C before it receives A's message

Channel States: A's transfer message becomes recorded as in-transit on the channel from A to B

Local States: A will have a balance of \$5 but B will still have a balance of \$10 since it has not received the message yet. C and D will also have a balance of \$10.

6 Teams

Projects can be done individually or in a team of 2.

7 Deadlines and Deployment

This project will be due Monday 02/07/2022. We will have a short demo for each project For this project's demo via Zoom (sign-up sheet for demo time slots and zoom link will be posted on piazza). You can deploy your code on several machines. However it is also acceptable if you just use several processes in the same machine to simulate the distributed environment.