Winter 2023 CS271 Project 2

January 27, 2023

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 token-passing application. There will be a single token that the clients will pass between each other. **Every client will be capable of initiating a global snapshot using the Chandy-Lamport algorithm.**

1 Application Component

We will assume 5 clients. One of them will start with a token. After holding onto the token for 1 second, the client with the token will randomly choose one of its outgoing channels and send the token to the client at the other end. With a small probability, the token can be lost. This process will continue to repeat indefinitely.

Your network connections should look like the directed graph in Figure 1. From looking at the figure, we can see that A can send to B and receive messages from B, but C can only send messages to (but cannot receive from) B. We can see that no messages can directly be exchanged between A and C.

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. If the token is lost, it will not show up in the snapshot, and the user can simply restart token-passing by giving the client a new token.

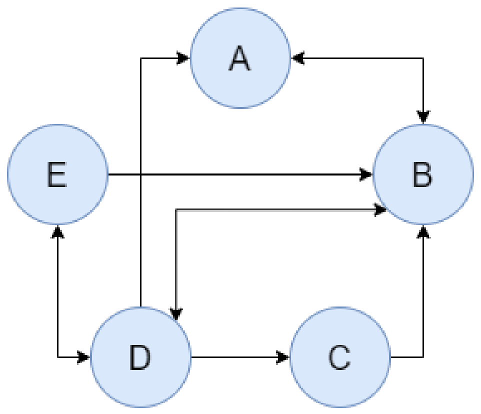

Figure 1: Network Connectivity Directed Graph for 5 Clients. A connection with double arrows represents two unidirectional channels back and forth between the two clients

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 preserve happens-before relationships. In other words, if event e happens-before event f, then in order 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 send and receive the token during the snapshot).
- Any process can initiate a global snapshot.

The following 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 MARKER messages on all outgoing channels

4. P_i starts recording incoming messages from all 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 MARKER messages on all outgoing channels
    4. P_j starts recording incoming messages from all incoming channels

- If P_j has 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 received on C_{kj} between the times 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 the channel states for all of its incoming channels to P_i, the process that initiated the snapshot

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 (token passing) messages.

3 User Interface

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

2. Through the client user interface, we can give that client the token so that it can start the process of token-passing.

3. Through the client user interface, we can issue a request to a specific client to start the snapshot algorithm. Once the snapshot has been taken, you should print the local states of all of the clients (i.e. whether or not a client has the token) as well as any messages in any of the channels between them (e.g. messages containing a token)

4. Through the client user interface, we can toggle that client to have a chance of losing the token when it receives the token. We should be able to specify the probability (e.g. 10% chance) of that client losing the token via this interface.

5. 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 sends or receives a token, print out whether the client has the token. Similarly, when a MARKER is received, print it out.

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

7. Use message passing primitives TCP/UDP. You can decide which alternative and explore the trade-offs. We will be interested in hearing about your experience. If you would like to use an alternative message-passing mechanism, please contact the teaching staff in advance.

4 Demo Case

For the demo, you should have 5 clients. Initially, none of the clients will have the token. All clients should start with a 0% chance of losing the token. Through the client user interface, we will give one of the clients a token to start passing to other clients. Once token-passing starts, we will pick a client to initiate the Global Snapshots protocol.
Here is an example demo case:

1. C is asked to initiate a snapshot

2. 1 second later A sends the token to B

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

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

Local States: Neither A or B will have the token since B has not received the token message yet. C and D also will not have the token.

5 Teams

This project can be done individually or in a team of 2.

6 Deadlines and Deployment

This project is due Friday 02/10/2023. We will have a short demo for each project The demo will be conducted via Zoom (signup sheet for demo time slots and the zoom link will be posted on Piazza). You can deploy your code on several machines. However, it is also acceptable if you use several processes in the same machine to simulate the distributed environment.

7 Extra Credit (15%)

Provide support for multiple clients to issue snapshots concurrently. In this case, MARKERS will need to carry unique snapshot IDs so MARKERS from different snapshots do not interfere with each other. Clients can assign globally unique IDs to each snapshot by concatenating their process ID and a local counter (e.g. *process_id.counter*). A snapshot should record other snapshots' MARKER messages as part of the state of the distributed system.