Chapter 4: Global State and Snapshot Recording Algorithms

Ajay Kshemkalyani and Mukesh Singhal

Distributed Computing: Principles, Algorithms, and Systems

Cambridge University Press

Introduction

- Recording the global state of a distributed system on-the-fly is an important paradigm.
- The lack of globally shared memory, global clock and unpredictable message delays in a distributed system make this problem non-trivial.
- This chapter first defines consistent global states and discusses issues to be addressed to compute consistent distributed snapshots.
- Then several algorithms to determine on-the-fly such snapshots are presented for several types of networks.

System model

- The system consists of a collection of n processes p_1, p_2, ..., p_n that are connected by channels.
- There are no globally shared memory and physical global clock and processes communicate by passing messages through communication channels.
- C_{ij} denotes the channel from process p_i to process p_j and its state is denoted by SC_{ij}.
- The actions performed by a process are modeled as three types of events: Internal events,the message send event and the message receive event.
- For a message m_{ij} that is sent by process p_i to process p_j, let $send(m_{ij})$ and $rec(m_{ij})$ denote its send and receive events.

System model

- At any instant, the state of process p_i, denoted by LS_i, is a result of the sequence of all the events executed by p_i till that instant.
- For an event e and a process state LS_i, $e \in LS_i$ iff e belongs to the sequence of events that have taken process p_i to state LS_i.
- For an event e and a process state LS_i, $e \notin LS_i$ iff e does not belong to the sequence of events that have taken process p_i to state LS_i.
- For a channel C_{ij}, the following set of messages can be defined based on the local states of the processes p_i and p_j

Transit: $transit(LS_i, LS_j) = \{ m_{ij} \mid send(m_{ij}) \in LS_i \bigwedge rec(m_{ij}) \notin LS_j \}$

Models of communication

Recall, there are three models of communication: FIFO, non-FIFO, and Co.

- In FIFO model, each channel acts as a first-in first-out message queue and thus, message ordering is preserved by a channel.
- In non-FIFO model, a channel acts like a set in which the sender process adds messages and the receiver process removes messages from it in a random order.
- A system that supports causal delivery of messages satisfies the following property: "For any two messages m_{ij} and m_{kj}, if $send(m_{ij}) \longrightarrow send(m_{kj})$, then $rec(m_{ij}) \longrightarrow rec(m_{kj})$".

Consistent global state

- The global state of a distributed system is a collection of the local states of the processes and the channels.
- Notationally, global state GS is defined as,

$$GS = \{\bigcup_i LS_i, \bigcup_{i,j} SC_{ij}\}$$

- A global state GS is a *consistent global state* iff it satisfies the following two conditions :

C1: $\text{send}(m_{ij}) \in LS_i \Rightarrow m_{ij} \in SC_{ij} \oplus \text{rec}(m_{ij}) \in LS_j$. ($\oplus$ is Ex-OR operator.)

C2: $\text{send}(m_{ij}) \notin LS_i \Rightarrow m_{ij} \notin SC_{ij} \wedge \text{rec}(m_{ij}) \notin LS_j$.

Interpretation in terms of cuts

- A cut in a space-time diagram is a line joining an arbitrary point on each process line that slices the space-time diagram into a PAST and a FUTURE.
- A consistent global state corresponds to a cut in which every message received in the PAST of the cut was sent in the PAST of that cut.
- Such a cut is known as a *consistent cut*.
- For example, consider the space-time diagram for the computation illustrated in Figure 4.1.
- Cut C1 is inconsistent because message m1 is flowing from the FUTURE to the PAST.
- Cut C2 is consistent and message m4 must be captured in the state of channel C_{21}.

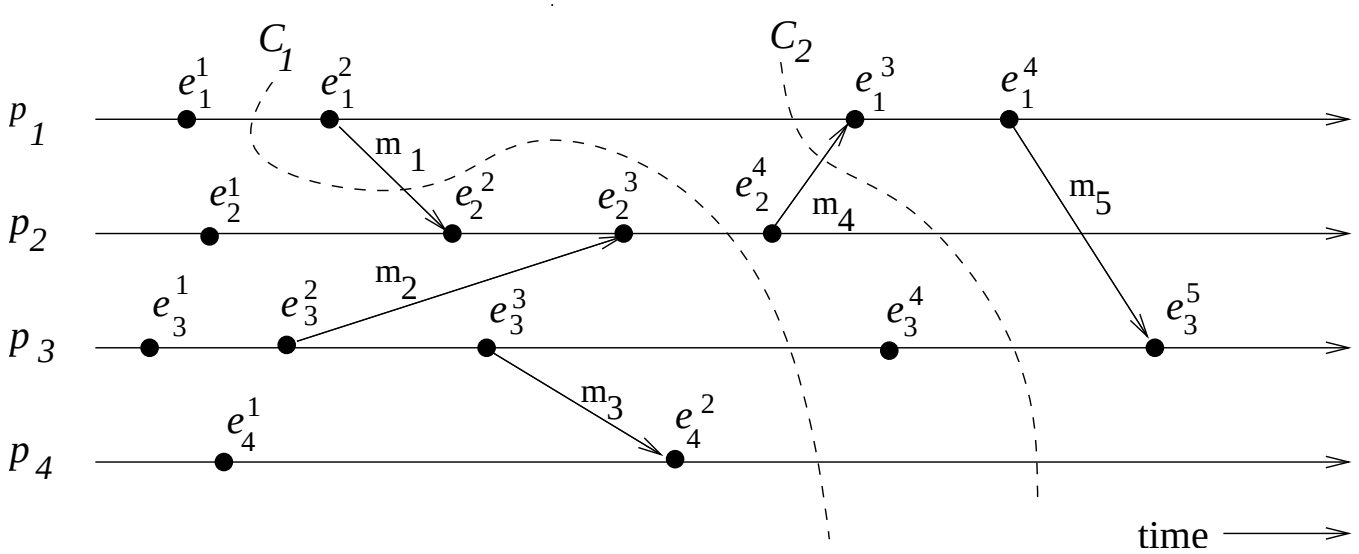

Figure 4.1: An Interpretation in Terms of a Cut.

Issues in recording a global state

The following two issues need to be addressed:

I1: How to distinguish between the messages to be recorded in the snapshot from those not to be recorded.

-Any message that is sent by a process before recording its snapshot, must be recorded in the global snapshot (from **C1**).
-Any message that is sent by a process after recording its snapshot, must not be recorded in the global snapshot (from **C2**).

I2: How to determine the instant when a process takes its snapshot.

-A process p_j must record its snapshot before processing a message m_{ij} that was sent by process p_i after recording its snapshot.

Snapshot algorithms for FIFO channels

Chandy-Lamport algorithm

- The Chandy-Lamport algorithm uses a control message, called a *marker* whose role in a FIFO system is to separate messages in the channels.
- After a site has recorded its snapshot, it sends a *marker*, along all of its outgoing channels before sending out any more messages.
- A marker separates the messages in the channel into those to be included in the snapshot from those not to be recorded in the snapshot.
- A process must record its snapshot no later than when it receives a marker on any of its incoming channels.

Chandy-Lamport algorithm

- The algorithm can be initiated by any process by executing the "Marker Sending Rule" by which it records its local state and sends a marker on each outgoing channel.
- A process executes the "Marker Receiving Rule" on receiving a marker. If the process has not yet recorded its local state, it records the state of the channel on which the marker is received as empty and executes the "Marker Sending Rule" to record its local state.
- The algorithm terminates after each process has received a marker on all of its incoming channels.
- All the local snapshots get disseminated to all other processes and all the processes can determine the global state.

Chandy-Lamport algorithm

Marker Sending Rule for process i

1. Process i records its state.
2. For each outgoing channel C on which a marker has not been sent, i sends a marker along C before i sends further messages along C.

Marker Receiving Rule for process j

On receiving a marker along channel C:

if j has not recorded its state **then**

Record the state of C as the empty set

Follow the "Marker Sending Rule"

else

Record the state of C as the set of messages received along C after j's state was recorded and before j received the marker along C