CSE535 Asynchronous Systems
Global State and Snapshot Recording

YoungMin Kwon

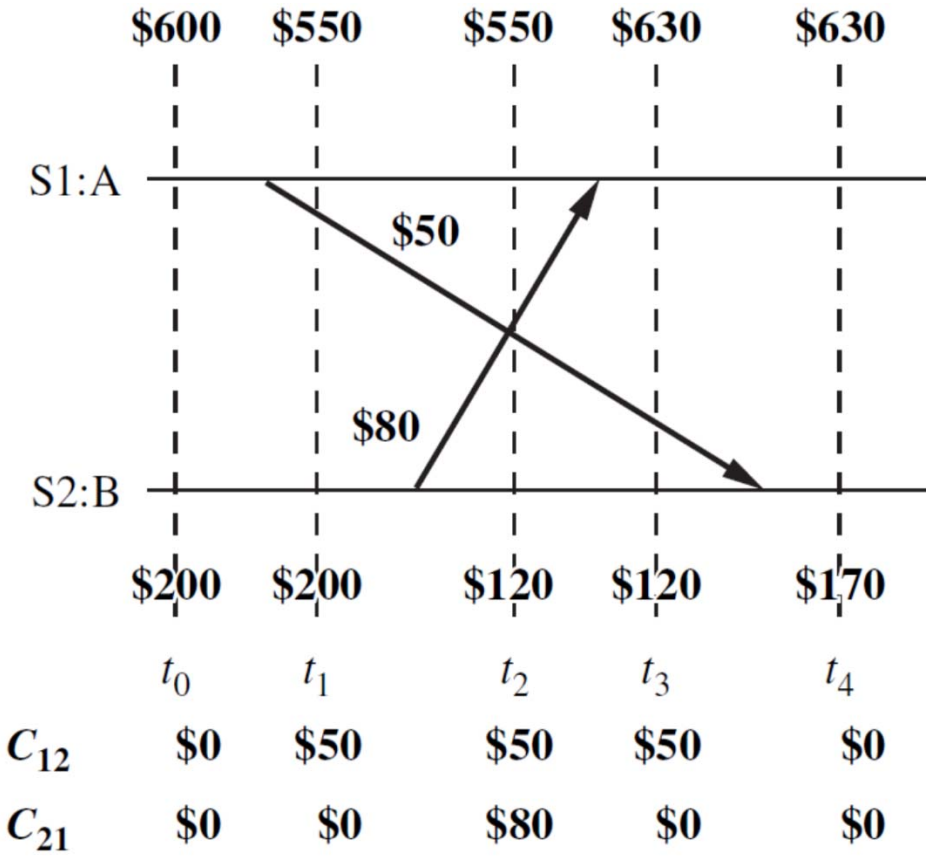
Global State Recording

- Global State:
  - States of processes + states of channels
  - Without shared memory or global clocks, it is difficult to obtain a global state.

- Uses of Global States:
  - Detection of properties like deadlocks, termination…
  - Failure recovery
  - Debugging distributed software…

Global State Recording

- A banking example:
  - S1 records the state at t_0
  - S2, C_{12}, C_{21} records states at t_2
  - global state has $850 in total



System Model

- The system consists of n processes p_1, p_2, …, p_n connected by channels
- Message send and receive is asynchronous with finite but arbitrary time delay.

- LS_i: the state of process p_i at a time instant
  - The set of events up to that time instant.
  - $e \in LS_i$ iff e is one of the events that have taken p_i to the state LS_i
- SC_{ij}: the state of the channel between p_i and p_j
  - The set of messages in transit in C_{ij}
  - transit(LS_i,LS_j): {m_{ij}: send(m_{ij}) $\in$ LS_i and rec(m_{ij}) $\notin$ LS_j}

System Model

- Model of communication
  - FIFO: each channel acts as a First-In First-Out queue.
  - Non-FIFO: channels act like a set where messages can be added and randomly removed.
  - Causal delivery: for any two messages m_{ij} and m_{kj} if $send(m_{ij}) \rightarrow send(m_{kj})$, then $rec(m_{ij}) \rightarrow rec(m_{kj})$

Consistent Global State

- Global state GS is defined as
  - GS = { $\cup_i LS_i$, $\cup_{i,j} SC_{ij}$ }

- A global state GS is a consistent global state iff

$$\textbf{C1: } send(m_{ij}) \in LS_i \Rightarrow m_{ij} \in SC_{ij} \oplus rec(m_{ij}) \in LS_j$$
$$\textbf{C2: } send(m_{ij}) \notin LS_i \Rightarrow m_{ij} \notin SC_{ij} \wedge rec(m_{ij}) \notin LS_j$$

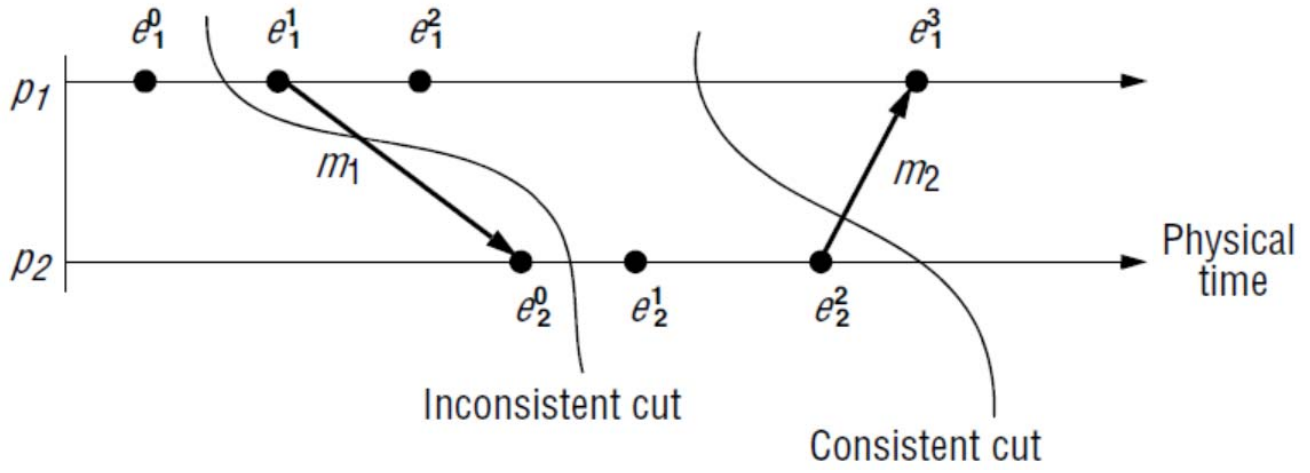
  - C1: conservation of messages.
  - C2: for every effect, its cause must be present.

Cuts

- A cut is a line joining an arbitrary point on each process line in the space-time diagram.
  - Events to the left of a cut are past events and to the right of the cut are future events.
  - Every cut represents a global state and every global state can be represented by a cut

Cuts

- Consistent cut
  - A cut such that every received message in the past is sent in the past.
  - A consistent cut corresponds to a consistent global state



Issues in Recording a Global State

- Distinguish the messages to be recorded in the snapshot and those not to be recorded

- Determine the instant when a process takes its snapshot.

Chandy-Lamport algorithm

- A snapshot algorithm for FIFO channels

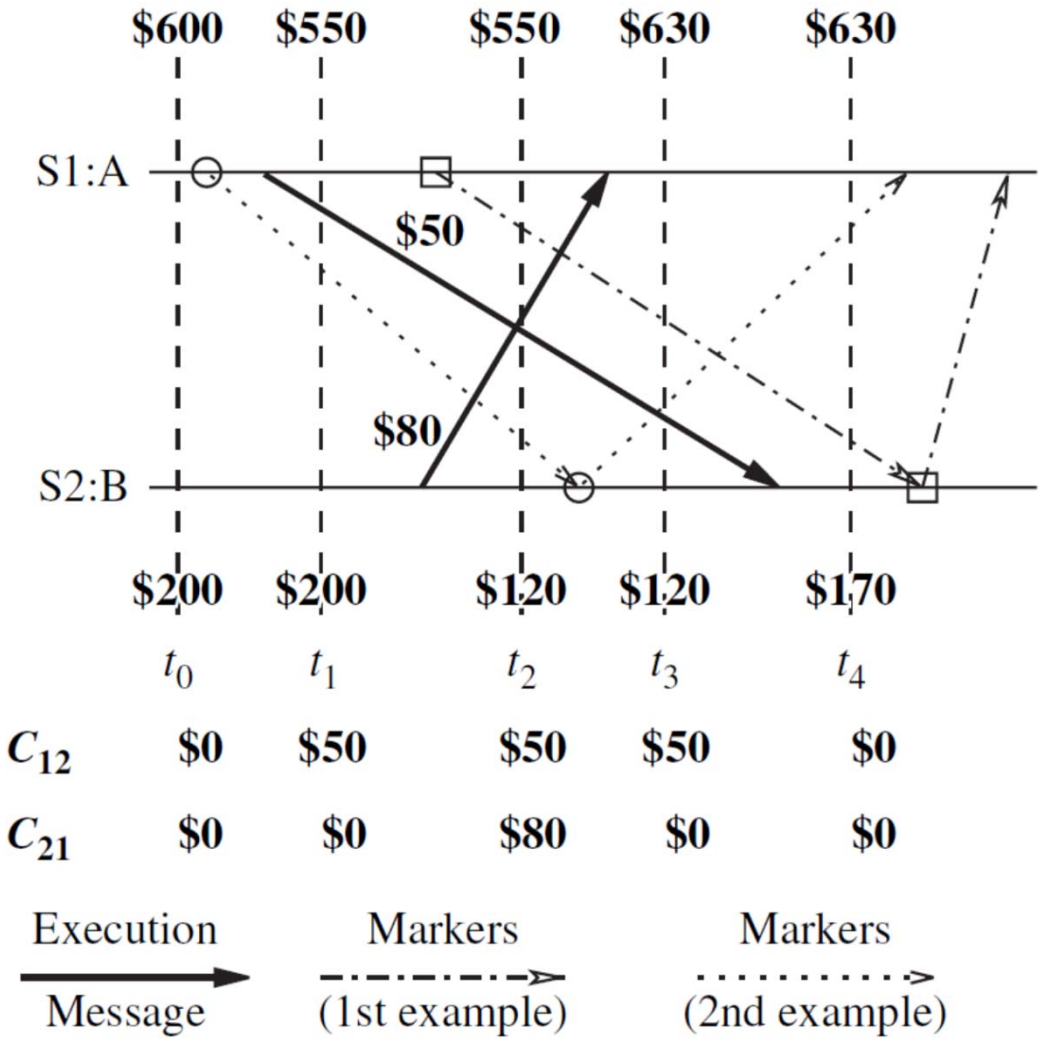
- Marker messages
  - After recoding a message, the process sends a marker message along all of its outgoing channels.
  - It distinguishes the messages sent before recording the state and those sent afterwards.
  - It tells the receiver to record its process state and its incoming channel states.

Chandy-Lamport algorithm

- Marker sending rule for p_i
  - p_i records its state
  - Send a marker along its outgoing channels

- Marker receiving rule for p_j (along a channel C)
  - If p_j has not recorded its state
    - Record the state of C as the empty set
    - Execute the "Marker sending Rule"
  - Else
    - Add to the state of C all messages received since p_j recorded its process state

Chandy-Lamport algorithm



Chandy-Lamport algorithm (Correctness)

- C1: $send(m_{ij}) \in LS_i \Rightarrow m_{ij} \in SC_{ij} \oplus rec(m_{ij}) \in LS_j$

  When p_j receives m_{ij},

  - If p_j has not taken a snapshot yet, m_{ij} will be in LS_i
  - Otherwise, m_{ij} will be in SC_{ij}

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

  - Due to the FIFO properties of channels, m_{ij} sent after marker is not in SC_{ij} and not in LS_j

Spezialetti-Kearns algorithm

- Two phases in obtaining a global snapshot
  - Locally recording the snapshot at every process
  - Distributing the resultant snapshot to the initiators
- Optimization from Chandy-Lamport algorithm
  - Concurrent initiation of snapshot collection
  - Efficient distribution of recorded snapshots
- Assuming bi-directional channels

Spezialetti-Kearns algorithm

- Master variable
  - Marker carries the initiator id of the algorithm
  - When p_j records its process state, p_j sets its master variable with the initiator id.
  - The initiator sets its master variable with its id.

- Region
  - All processes whose master variable contains the same initiator id.

Spezialetti-Kearns algorithm:
Snapshot Recording

- Multiple initiators start the algorithm concurrently

- On receiving a marker,
  - If the process state has not been recorded, record it and update the master variable with the initiator id
  - Record the channel state like Chandy-Lamport algorithm.
  - Add the initiator id to its id-border-set.
  - If the master id of the marker is different than its master variable, do not distribute the marker.

Spezialetti-Kearns algorithm:
Dissemination of the recorded snapshot

- Implicit forest of spanning trees
  - Each initiator is the root of a spanning tree
  - If p_j executed the "marker sending rule," in response to p_i's marker message, p_i becomes the parent of p_j

Spezialetti-Kearns algorithm:
Dissemination of the recorded snapshot

- When a leaf node recorded all of its incoming channels, it sends its recorded states to its parent
- When an intermediate node recorded all of its incoming channels and all of its children's states, forward them to its parent.
- Initiators exchange new recorded states with other initiators in the id-border-set.

Venkatesan's Incremental Snapshot algorithm

- When snapshots need to be taken repeatedly
  - e.g. Synchronous checkpointing

- Assumption
  - Bidirectional FIFO channel
  - Single initiator
  - Fixed spanning tree in the network

Venkatesan's Incremental Snapshot algorithm

- Four types of message
  - init_snap, snap_completed:
    - Traverse the spanning tree edges
  - regular, ack:
    - To record non-spanning tree edges
    - Can be skipped if no messages were sent since the last snapshot

Venkatesan's Incremental Snapshot algorithm

- Algorithm
  - Snapshots are assigned a version number
  - The initiator sends init_snap message along the spanning tree edges
  - On receiving init_snap or regular messages with a new version number, the receiver process runs the marker sending rule

Venkatesan's Incremental Snapshot algorithm

- Marker sending rule
  - After recording its state, a process sends a regular message to the channels used since the last snapshot and waits for the ack message
  - Leaf processes: after receiving all expected ack messages, send snap_completed message to its parent
  - Non-leaf processes: after receiving all expected ack messages and snap_completed messages from all of its children, send snap_completed to its parent.