Global State Recording

Global State Collection

- Applications:
  – Checking "stable" properties, checkpoint & recovery

- Issues:
  – Need to capture both node and channel states
  – system cannot be stopped
  – no global clock

Notations

Some notations:

- LS_i: Local state of process i
- send(m_{ij}) : Send event of message m_{ij} from process i to process j
- rec(m_{ij}) : Similar, receive instead of send
- time(x) : Time at which state x was recorded
- time (send(m)) : Time at which send(m) occurred

Definitions

- $send(m_{ij}) \in LS_i$ iff $time(send(m_{ij})) < time(LS_i)$
- $rec(m_{ij}) \in LS_j$ iff $time(rec(m_{ij})) < time(LS_j)$
- $transit(LS_i, LS_j)$

  $= \{ m_{ij} \mid send(m_{ij}) \in LS_i \text{ and } rec(m_{ij}) \notin LS_j \}$
- $inconsistent(LS_i, LS_j)$

  $= \{ m_{ij} \mid send(m_{ij}) \notin LS_i \text{ and } rec(m_{ij}) \in LS_j \}$

Definitions

- Global state: collection of local states

  GS = {LS1, LS2,…, LSn}

- GS is consistent iff

  for all i, j, $1 \leq i, j \leq n$,

  inconsistent(LSi, LSj) = Φ

- GS is transitless iff

  for all i, j, $1 \leq i, j \leq n$,

  transit(LSi, LSj) = Φ

- GS is strongly consistent if it is consistent and transitless.

Chandy-Lamport’s Algorithm

- Uses special marker messages.

- One process acts as initiator, starts the state collection by following the marker sending rule below.

- Marker sending rule for process P:
  - P records its state and
  - For each outgoing channel C from P on which a marker has not been sent already, P sends a marker along C before any further message is sent on C

Chandy Lamport’s Algorithm contd..

- When Q receives a marker along a channel C:
  - If Q has not recorded its state then Q records the state of C as empty; Q then follows the marker sending rule
  - If Q has already recorded its state, it records the state of C as the sequence of messages received along C after Q’s state was recorded and before Q received the marker along C

Notable Points

- Markers sent on a channel distinguish messages sent on the channel before the sender recorded its states and the messages sent after the sender recorded its state
- The state collected may not be any state that actually happened in reality, rather a state that "could have" happened
- Requires FIFO channels
- Message complexity $O(|E|)$, where E = no. of links

Termination Detection

Termination Detection

- Model
  - processes can be active or idle
  - only active processes send messages
  - idle process can become active on receiving a computation message
  - active process can become idle at any time

  - Termination: all processes are idle and no computation message are in transit
  - Can use global snapshot to detect termination also

Huang's Algorithm

- One controlling agent, has weight 1 initially
- All other processes are idle initially and has weight 0
- Computation starts when controlling agent sends a computation message to a process
- An idle process becomes active on receiving a computation message
- B(DW) – computation message with weight DW. Can be sent only by the controlling agent or an active process
- C(DW) – control message with weight DW, sent by active processes to controlling agent when they are about to become idle

Weight Distribution and Recovery

- Let current weight at process = W
- Send of B(DW):
  - Find W1, W2 such that W1 > 0, W2 > 0, W1 + W2 = W
  - Set W = W1 and send B(W2)
- Receive of B(DW):
  - W = W + DW;
  - if idle, become active
- Send of C(DW):
  - send C(W) to controlling agent
  - Become idle
- Receive of C(DW):
  - W = W + DW
  - if W = 1, declare "termination"