Part – C

1) **Spezialetti–Kearns algorithm:**

Spezialetti and Kearns provided two optimizations to the Chandy–Lamport algorithm. The first optimization combines snapshots concurrently initiated by multiple processes into a single snapshot. Second , deals with the efficient distribution of the global snapshot.

Efficient snapshot recording

- Every marker carries the identifier of the initiator
- Each process has a variable master to keep track of the initiator
- A process that initiates the algorithm records its own identifier in the master variable
- ***Region*** - encompasses all the processes whose master field contains the identifier of the same initiator
- When there are multiple concurrent initiators, the system gets partitioned into multiple regions
- When the initiator's identifier in a marker received along a channel is different from the value in the master variable, a concurrent initiation of the algorithm is detected and the sender of the marker lies in a different region
- The identifier of the concurrent initiator is recorded in a local variable ***id-border-set***
- The process receiving the marker does not take a snapshot for this marker and does not propagate this marker
- A process does not take a snapshot or propagate a snapshot request initiated by a process if it has already taken a snapshot in response to some other snapshot initiation
- snapshot recorded in one region will be merged with the snapshot recorded in the adjacent region. Because, they are considered part of the same instance of the algorithm for the purpose of channel state recording.
- Snapshot recording at a process is complete after it has received a marker along each of its channels
- ***id-border-set*** - contains the identifiers of the neighbouring regions

Efficient dissemination of the recorded snapshot

- The initiator of the algorithm is the root of a spanning tree and all processes in its region belong to its spanning tree
- If process pi executed the "marker sending rule" because it received its first marker from process pj, then process pj is the parent of process pi in the spanning tree
- When a leaf process in the spanning tree has recorded the states of all incoming channels, the process sends the locally recorded state. (Intermediate will receive → forward to its parent)
- When the initiator receives it, it assembles the snapshot for all the processes in its region and the channels incident on these processes
- . The initiator exchanges the snapshot of its region with the initiators in adjacent regions in rounds, with the help of ***id-border-set***
- A round is complete when an initiator receives information, or the blank message from all initiators of adjacent regions from which it has not already received a blank message (no update)

2) Venkatesan’s incremental snapshot algorithm

- Repeated collection of global snapshots will make the overall system to be in a consistent state
- In this approach a step is there to record an new snapshot after the most recent snapshot was taken and combine it with the new snapshot to obtain the latest snapshot of the system
- Thus does a process of saving the communication when computation messages are sent only on a few of the network channels, between the recording of two successive snapshots
- Parameters Involved: single initiator, a fixed spanning tree in the network, and four types of control messages: ***init_snap, regular, and ack. init_snap, and snap_completed***
- ***regular and ack messages***, which serve to record the state of non-spanning edges, are not sent on those edges on which no computation message has been sent since the previous snapshot
- Each snapshot will have an version number in it
- The initiator notifies all the processes about the version number of the new snapshot by sending init_snap messages along the spanning tree edges
- A process follows the “marker sending rule” when it receives this notification or when it receives a regular message with a new version number
- The “marker sending rule” is modified so that the process sends regular messages along only those channels on which it has sent computation messages since the previous snapshot, and the process waits for ack messages in response to these regular messages
- When a leaf process in the spanning tree receives all the ack messages it expects, it sends a snap_completed message to its parent process
- When a non-leaf process in the spanning tree receives all the ack messages it expects, as well as a snap_completed message from each of its child processes, it sends a snap_completed message to its parent process
- The algorithm terminates when the initiator has received all the ack messages it expects, as well as a snap_completed message from each of its child processes

3) Helary’s wave synchronization method

- A wave is a flow of control messages such that every process in the system is visited exactly once by a wave control message, and at least one process in the system can determine when this flow of control messages terminates
- A wave is initiated after the previous wave terminates
- A “marker sending rule” is executed when a control message belonging to the wave flow visits the process. The process then forwards a control message to other processes, depending on the wave traversal structure, to continue the wave’s progression
- The “marker receiving rule” is modified so that if the process has not recorded its state when a marker is received on some channel, the “marker receiving rule” is not executed and no messages received after the marker on this channel are processed until the control message belonging to the wave flow visits the process
- Thus, each process follows the “marker receiving rule” only after it is visited by a control message belonging to the wave

(i.e.): number of messages in transit to each process in a global snapshot, and whether the global snapshot is strongly consistent

Workflow Implementation:

- Each process maintains two vectors, SENT and RECD.
- The ith elements of these vectors indicate the number of messages sent to/received from process i, respectively, since the previous visit of a wave control message
- The wave control messages carry a global abstract counter vector whose ith entry indicates the number of messages in transit to process i
- These entries in the vector are updated using the SENT and RECD vectors at each node visited
- When the control wave terminates, the number of messages in transit to each process as recorded in the snapshot is known