TOTAL ORDER

For each pair of processes P_i and P_j and for each pair of messages M_x and M_y that are delivered to both the processes, P_i is delivered M_x before M_y if and only if P_j is delivered $M_x$before M_y.

Centralized Algorithm for total ordering

Each process sends the message it wants to broadcast to a centralized process, whichrelays all the messages it receives to every other process over FIFO channels.

(1) When process Pi wants to multicasts a message M to group G: (1a) send M(i, G) to central coordinator

(2) When M(i, G) arrives from P_i at the central coordinator (2a) send M(i, G) to all members of the group G.

(3) When M(i, G) arrives at p_j from the central coordinator (3a) deliver M(i, G) to the application.

Complexity: Each message transmission takes two message hops and exactly n messages in asystem of n processes.

Drawbacks: A centralized algorithm has a single point of failure and congestion, and is not an elegant solution.

Three phase distributed algorithm

Three phases can be seen in both sender and receiver side.

Sender side

Phase 1

- In the first phase, a process multicasts the message M with a locally unique tag and thelocal timestamp to the group members.

Phase 2

- The sender process awaits a reply from all the group members who respond with atentative proposal for a revised timestamp for that message M.

- The await call is non-blocking.

Phase 3

- The process multicasts the final timestamp to the group.

```
Record Q_entry
        M: int;                         // the application message
        tag: int;                       // unique message identifier
        sender_id: int;                 // sender of the message
        timestamp: int;         // tentative timestamp assigned to message
        deliverable: Boolean    // whether message is ready for delivery
(local variables)
queue of Q_entry: temp_Q, delivery_Q
int: clock              // Used as a variant of Lamport's scalar clock
int: priority           // Used to track the highest proposed timestamp (message types)
REVISE_TS(M, i, tag, ts)
                        // Phase 1 message sent by Pi, with initial timestamp ts
PROPOSED_TS(j, i, tag, ts)
                        // Phase 2 message sent by Pj, with revised timestamp Pi
FINAL_TS(i, tag, ts) // Phase 3 message sent by Pi, with final timestamp
(1)   When process Pi wants to multicast a message M with a tag tag:
(1a) clock ← clock + 1;
(1b) send REVISE_TS(M, I, tag, clock) to all processes;
(1c) temp_ts ← 0
(1d) await PROPOSED_TS(j, i, tag, tsj) from each process Pj;
(1e) ∀ j ∈ N , do temp_ts ← max(temp_ts, tsj);
(1f) send FINAL_TS(i, tag, temp_ts) to all processes;
(1g) clock ← max(clock, temp_ts)
```

Fig: Sender side of three phase distributed algorithm

Receiver Side

Phase 1

- The receiver receives the message with a tentative timestamp. It updates the variable priority that tracks the highest proposed timestamp, then revises the proposed timestamp to the priority, and places the message with its tag and the revisedtimestamp at the tail of the queue temp_Q. In the queue, the entry is marked as undeliverable.

Phase 2

- The receiver sends the revised timestamp back to the sender. The receiver then waitsin a non-blocking manner for the final timestamp.

Phase 3

- The final timestamp is received from the multicaster. The corresponding message entry in temp_Q is identified using the tag, and is marked as deliverable after the revised timestamp is overwritten by the final timestamp.
- The queue is then resorted using the timestamp field of the entries as the key. As the queue is already sorted except for the modified entry for the message under consideration, that message entry has to be placed in its sorted position in the queue.
- If the message entry is at the head of the temp_Q, that entry, and all consecutive subsequent entries that are also marked as deliverable, are dequeued from temp_Q, and enqueued in deliver_Q.

Complexity

This algorithm uses three phases, and, to send a message to n − 1 processes, it uses 3(n - 1) messages and incurs a delay of three message hops