RAMAIAH
Institute of Technology

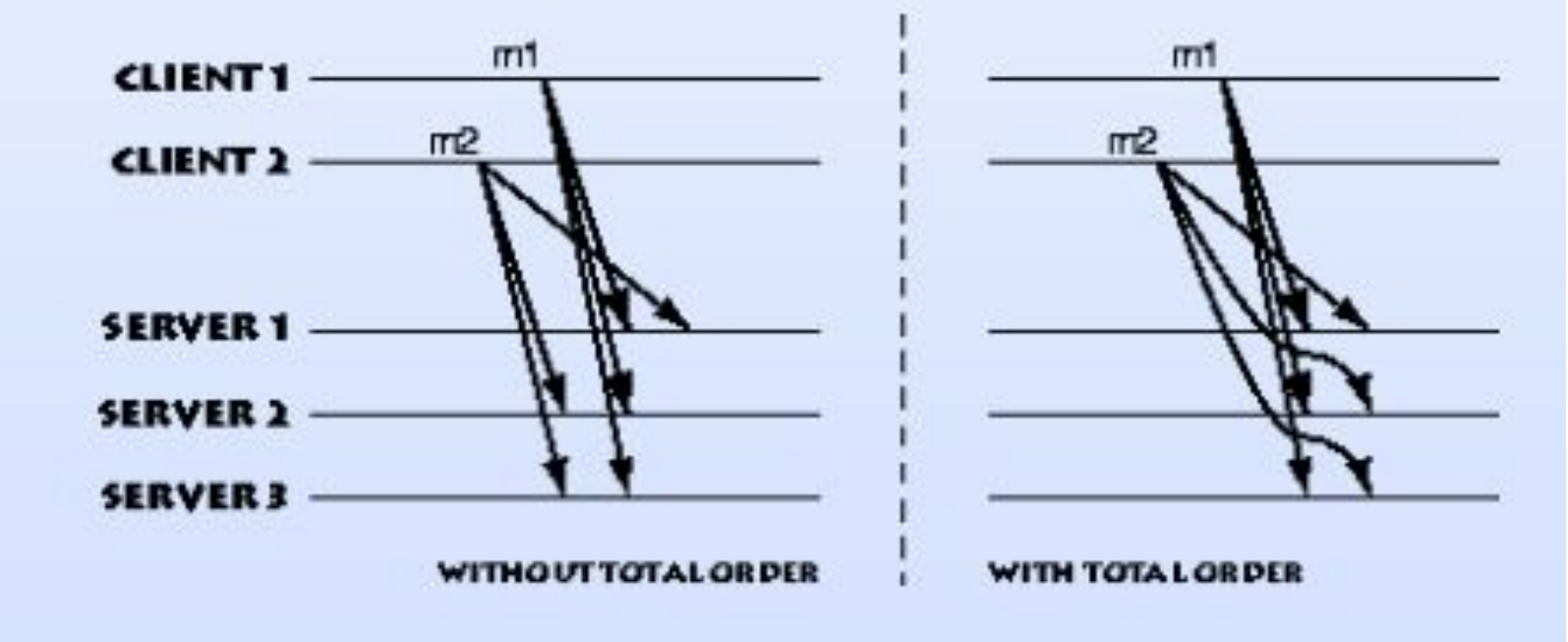
M.S. Ramaiah Institute of Technology
(Autonomous Institute, Affiliated to VTU)
Department of Computer Science and Engineering

Course Name: Distributed Systems
Course Code: CSE751/CSE20
Credits: 3:0:0/3:0:0:1

Term: Oct 2021– Feb 2022

Faculty:
Sini Anna Alex

Total Order


CLIENT 1
CLIENT 2
SERVER 1
SERVER 2
SERVER 3
m1
m2
WITHOUT TOTAL ORDER
m1
m2
WITH TOTAL ORDER

RAMAIAH
Institute of Technology

Centralized algorithm for total order

Assuming all processes broadcast messages, the centralized solution shown in Algorithm 6.4 enforces total order in a system with FIFO channels. Each process sends the message it wants to broadcast to a centralized process, which simply relays all the messages it receives to every other process over FIFO channels. It is straightforward to see that total order is satisfied. Furthermore, this algorithm also satisfies causal message order.

---

(1) When process P_i wants to multicast 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.

Algorithm 6.4 A centralized algorithm to implement total order and causal order of messages.

Complexity

Each message transmission takes two message hops and exactly n messages in a system of n processes.

Drawbacks

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

Three-phase distributed algorithm

Sender

Phase 1 In the first phase, a process multicasts (line 1b) the message M with a locally unique tag and the local timestamp to the group members.

Phase 2 In the second phase, the sender process awaits a reply from all the group members who respond with a tentative proposal for a revised timestamp for that message M. The **await** call in line 1d is non-blocking,

i.e., any other messages received in the meanwhile are processed. Once all expected replies are received, the process computes the maximum of the proposed timestamps for M, and uses the maximum as the final timestamp.

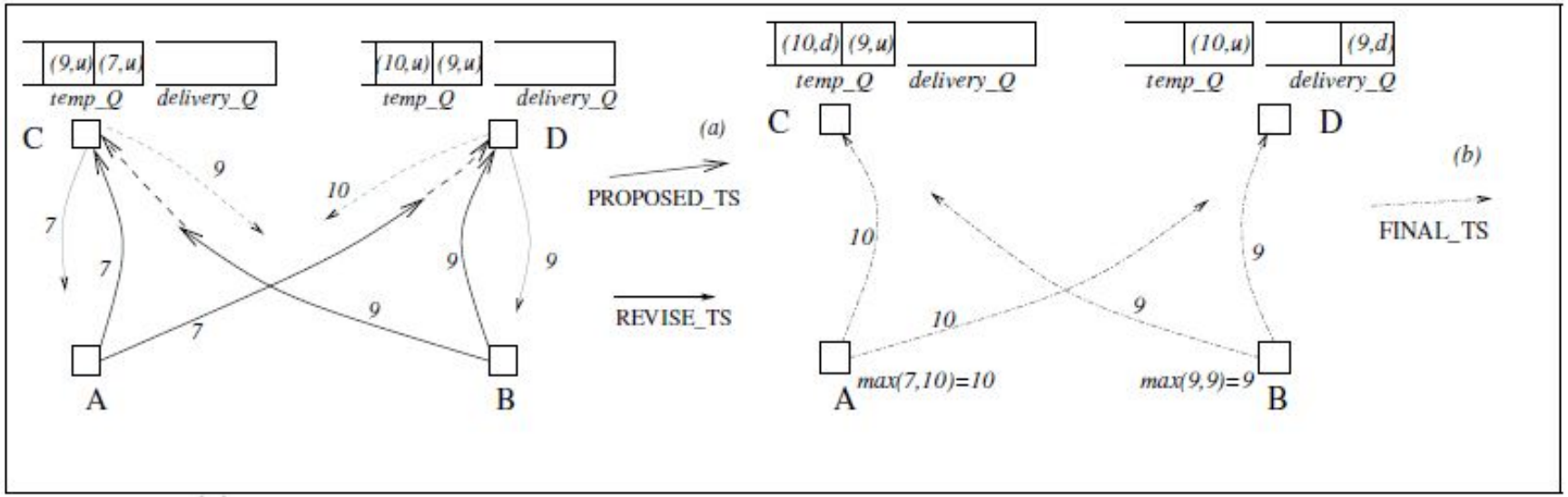
Phase 3 In the third phase, the process multicasts the final timestamp to the group in line (1f).

Receivers

Phase 1 In the first phase, the receiver receives the message with a tentative/proposed timestamp. It updates the variable *priority* that tracks the highest proposed timestamp (line 2a), then revises the proposed timestamp to the *priority*, and places the message with its tag and the revised timestamp at the tail of the queue *temp_Q* (line 2b). In the queue, the entry is marked as undeliverable.

Phase 2 In the second phase, the receiver sends the revised timestamp (and the tag) back to the sender (line 2c). The receiver then waits in a non-blocking manner for the final timestamp (correlated by the message tag).

Phase 3 In the third phase, the final timestamp is received from the multicaster (line 3). The corresponding message entry in *temp_Q* is identified using the tag (line 3a), and is marked as deliverable (line 3b) after the revised timestamp is overwritten by the final timestamp (line 3c). The queue is then resorted using the timestamp field of the entries as the key (line 3c). 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* in that order (the loop in lines 3d–3g).



The main sequence of steps is as follows:

1. A sends a *REVISE_TS*(7) message, having timestamp 7. B sends a *REVISE_TS*(9) message, having timestamp 9.

2. C receives A's *REVISE_TS*(7), enters the corresponding message in *temp_Q*, and marks it as undeliverable; *priority* = 7. C then sends *PROPOSED_TS*(7) message to A.

3. D receives B's REVISE_TS(9), enters the corresponding message in temp_Q, and marks it as undeliverable; priority = 9. D then sends PROPOSED_TS(9) message to B.

4. C receives B's *REVISE_TS*(9), enters the corresponding message in *temp_Q*, and marks it as undeliverable; *priority* = 9. C then sends *PROPOSED_TS*(9) message to B.

5. D receives A's *REVISE_TS*(7), enters the corresponding message in *temp_Q*, and marks it as undeliverable; *priority* = 10. D assigns a tentative timestamp value of 10, which is greater than all of the timestamps on *REVISE_TS* seen so far, and then sends *PROPOSED_TS*(10) message to A.

The continuing sequence of main steps is as follows:

6. When A receives *PROPOSED_TS*(7) from C and *PROPOSED_TS*(10) from D, it computes the *FINAL_TS*(10) to C and D.

7. When B receives *PROPOSED_TS*(9) from C and *PROPOSED_TS*(9) from D, it computes the final timestamp as

max(9, 9) = 9, and sends *FINAL_TS*(9) to C and D.

8. C receives *FINAL_TS*(10) from A, updates the corresponding entry in *temp_Q* with the timestamp, resets the queue, and marks the message as deliverable. As the message is not at the head of the queue, and some entry ahead of it is still undeliverable, the message is not moved to *delivery_Q*.

9. D receives *FINAL_TS*(9) from B, updates the corresponding entry in *temp_Q* by marking the corresponding message as deliverable, and resets the queue. As the message is at the head of the queue, it is moved to *delivery_Q*. Mark timestamp as max(7, 10) = 10, and sends *FINAL_TS*(10) to C and D.

10. When C receives *FINAL_TS*(9) from B, it will update the corresponding entry in *temp_Q* by marking the corresponding message as deliverable. As the message is at the head of the queue, it is moved to the *delivery_Q*, and the next message (of A), which is also deliverable, is also moved to the *delivery_Q*.

11. When D receives *FINAL_TS*(10) from A, it will update the corresponding entry in *temp_Q* by marking the corresponding message as deliverable. As the message is at the head of the queue, it is moved to the *delivery_Q*.

Algorithm by Kshemkalyani–Singhal to optimally implement causal ordering

Multicasts M_{51} and M_{42}

Message M_{51} sent to processes P_4 and P_6 contains the piggybacked information "M_{51}Dests = P_4P_6."

Multicast $M_{4,3}$

At event (4, 3), the information "$P_6 \in M_{5,1}.Dests$" in Log_4 is propagated on multicast $M_{4,3}$ only to process P_6 to ensure causal delivery using the Delivery Condition. The piggybacked information on message $M_{4,3}$ sent to process P_3 must not contain this information because of constraint II. (The piggybacked information contains "$M_{4,3}.Dests = \{P_6\}$." As long as any future message

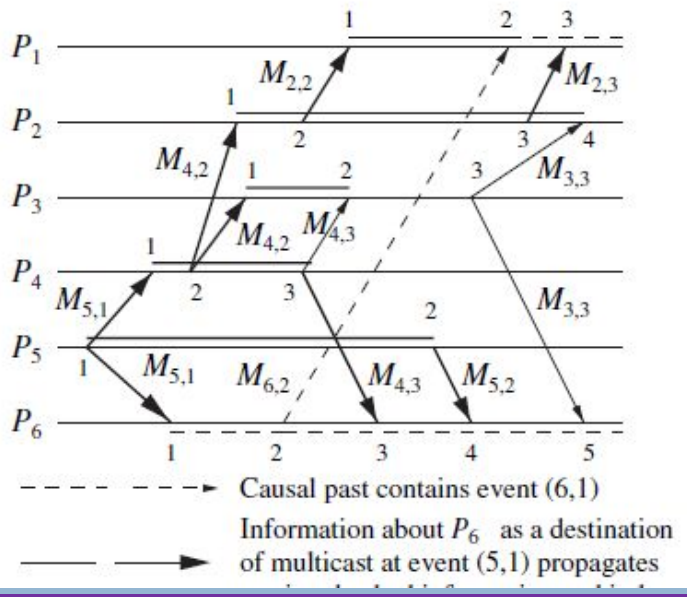


Figure 6.13 An example to illustrate the propagation constraints [6].

Message to dest.	Piggybacked $M_{5,1}$.Dests
$M_{5,1}$ to P_4, P_6	$\{P_4, P_6\}$
$M_{4,2}$ to P_3, P_2	$\{P_6\}$
$M_{2,2}$ to P_1	$\{P_6\}$
$M_{6,2}$ to P_1	$\{P_4\}$
$M_{4,3}$ to P_6	$\{P_6\}$
$M_{4,3}$ to P_3	$\{\}$
$M_{5,2}$ to P_6	$\{P_4, P_6\}$
$M_{2,3}$ to P_1	$\{P_6\}$
$M_{3,3}$ to $P_{2,6}$	$\{\}$

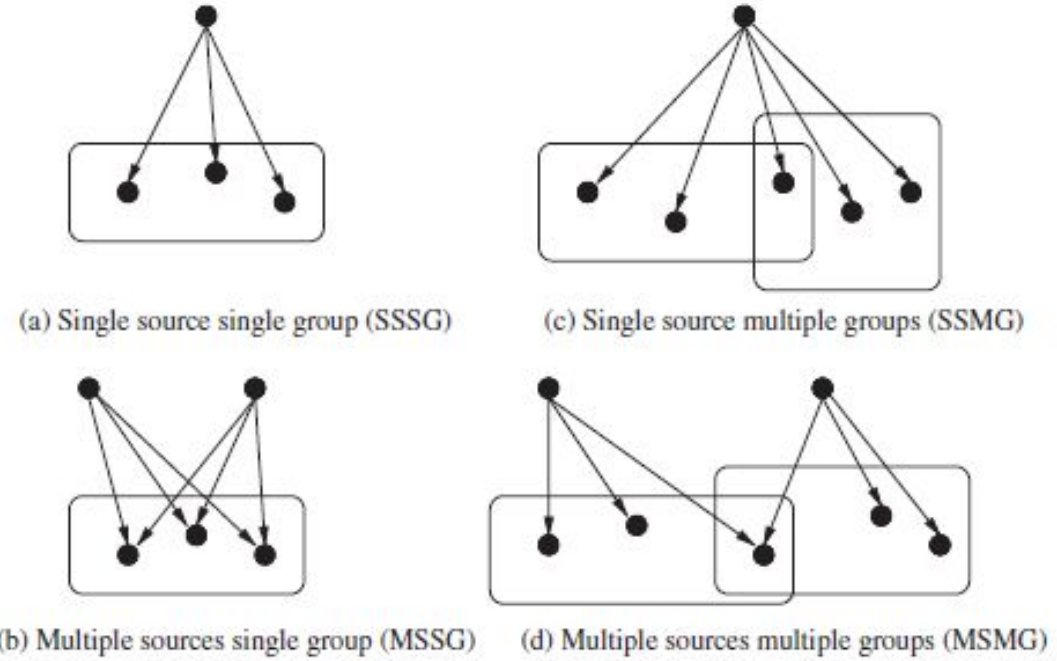
A nomenclature for multicast

- **SSSG** Single source and single destination group.
- **MSSG** Multiple sources and single destination group.
- **SSMG** Single source and multiple, possibly overlapping, groups.
- **MSMG** Multiple sources and multiple, possibly overlapping, groups.

The SSSG and SSMG classes are straightforward to implement, assuming the presence of FIFO channels between each pair of processes. Both total order and causal order are guaranteed.

The MSSG class is also straightforward to handle; the centralized implementation in Algorithm provides both total and causal order.
For the MSMG class, the approach, commonly termed as the *propagation tree* approach, uses a semi-centralized structure that adapts the centralized algorithm

Figure 6.15 Four classes of source–destination relationships for open-group multicasts. For closed-group multicasts, the sender needs to be part of the recipient group.



Thank you