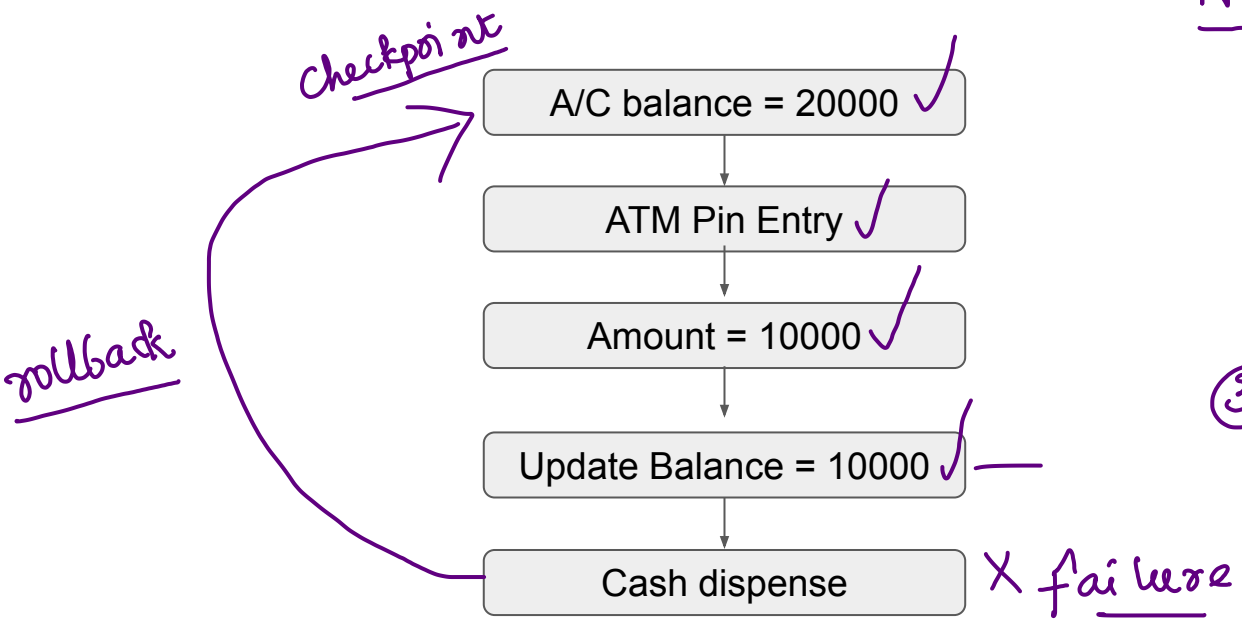
CS 3551 DISTRIBUTED COMPUTING

UNIT IV CONSENSUS AND RECOVERY **10**

Consensus and Agreement Algorithms: Problem Definition – Overview of Results – Agreement in a Failure-Free System(Synchronous and Asynchronous) – Agreement in Synchronous Systems with Failures; Checkpointing and Rollback Recovery: Introduction – Background and Definitions – Issues in Failure Recovery – Checkpoint-based Recovery – Coordinated Checkpointing Algorithm -- Algorithm for Asynchronous Checkpointing and Recovery

Checkpoint



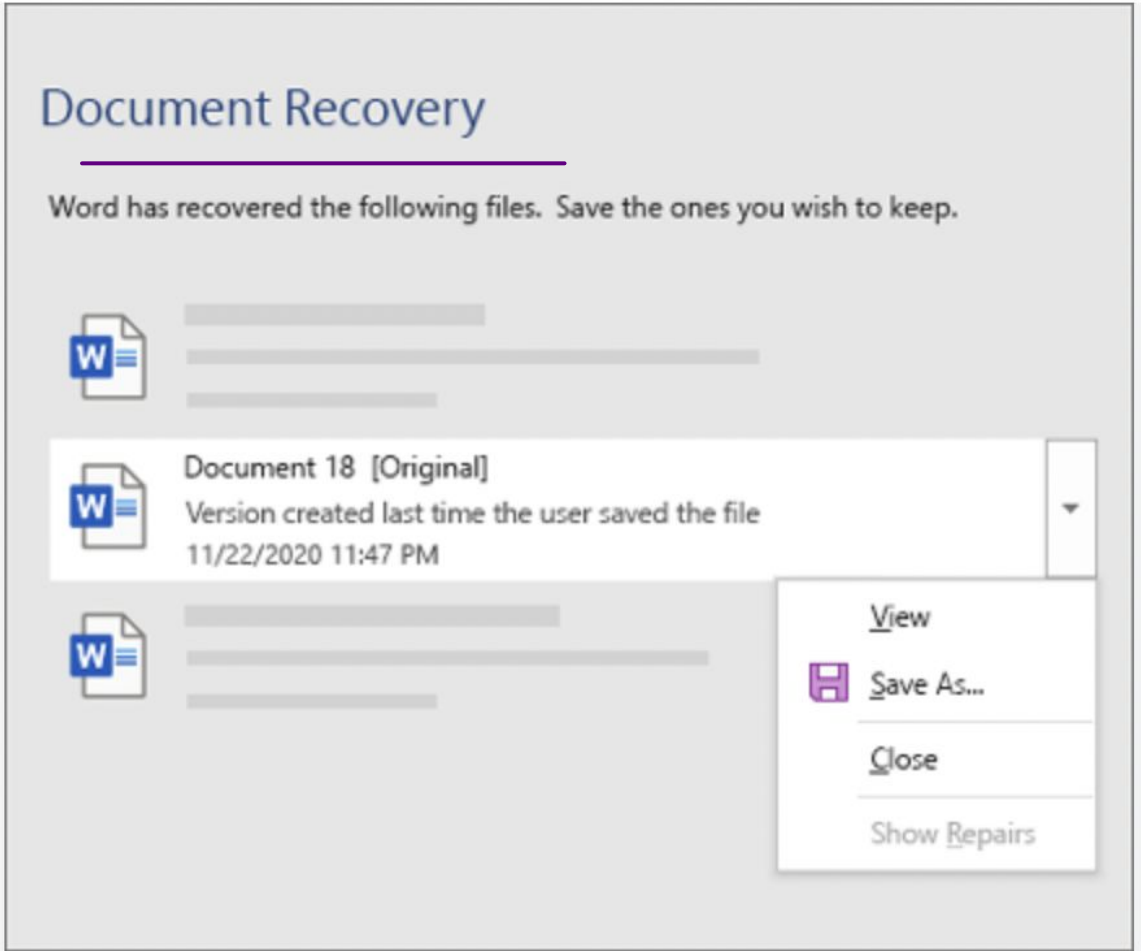
② NFS

L1 ✓
L2 ✓
L3 X

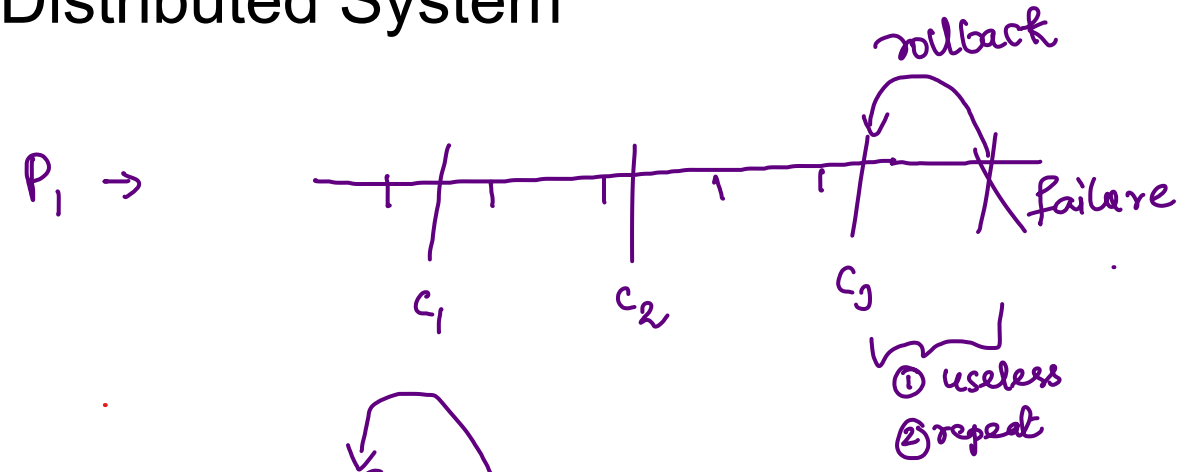
L10

③ Word

Microsoft Support

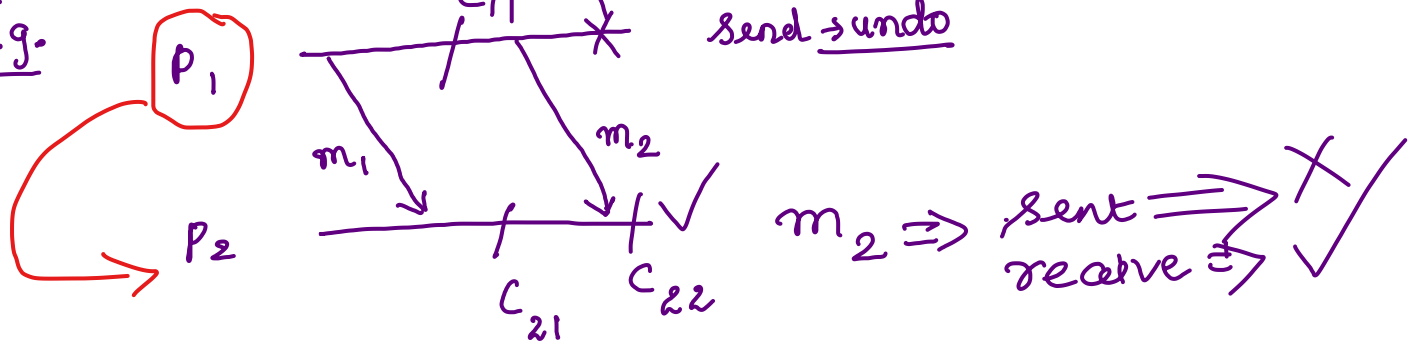


Checkpoint in Distributed System



domino

Eg.



What is Domino Effect?

- To see why rollback propagation occurs, consider the situation where the sender of a message m rolls back to a state that precedes the sending of m.
- The receiver of m must also roll back to a state that precedes m’s receipt; otherwise, the states of the two processes would be inconsistent because they would show that message m was received without being sent, which is impossible in any correct failure-free execution.
- This phenomenon of cascaded rollback is called the domino effect.
- In some situations, rollback propagation may extend back to the initial state of the computation, losing all the work performed before the failure.

Domino effect continued…

- **Independent or uncoordinated checkpointing** : - If each participating process takes its checkpoints independently, then the system is susceptible to the domino effect.

How to avoid domino effect?

1. **Coordinated checkpointing :**
   - processes coordinate their checkpoints to form a system-wide consistent state.
   - In case of a process failure, the system state can be restored to such a consistent set of checkpoints, preventing the rollback propagation.
2. **Communication-induced checkpointing :**
   - forces each process to take checkpoints based on information piggybacked on the application messages it receives from other processes.
   - Checkpoints are taken such that a system-wide consistent state always exists on stable storage, thereby avoiding the domino effect.
3. **Logbased rollback recovery:**
   - combines checkpointing with logging of nondeterministic events.
   - Log-based rollback recovery relies on the piecewise deterministic (PWD) assumption, which postulates that all non-deterministic events that a process executes can be identified and that the information necessary to replay each event during recovery can be logged in the event's determinant.
   - By logging and replaying the non-deterministic events in their exact original order, a process can deterministically recreate its pre-failure state even if this state has not been checkpointed.

Key Points

- Rollback recovery treats a distributed system application as a collection of processes that communicate over a network.
- It achieves fault tolerance by periodically saving the state of a process during the failure-free execution, enabling it to restart from a saved state upon a failure to reduce the amount of lost work.
- The saved state is called a checkpoint, and the procedure of restarting from a previously checkpointed state is called rollback recovery.
- A checkpoint can be saved on either the stable storage or the volatile storage depending on the failure scenarios to be tolerated.
- **Challenges for Recovery:**
    - on a failure of one or more processes in a system, these dependencies may force some of the processes that did not fail to roll back, creating what is commonly called a rollback propagation

LIKE

COMMENT

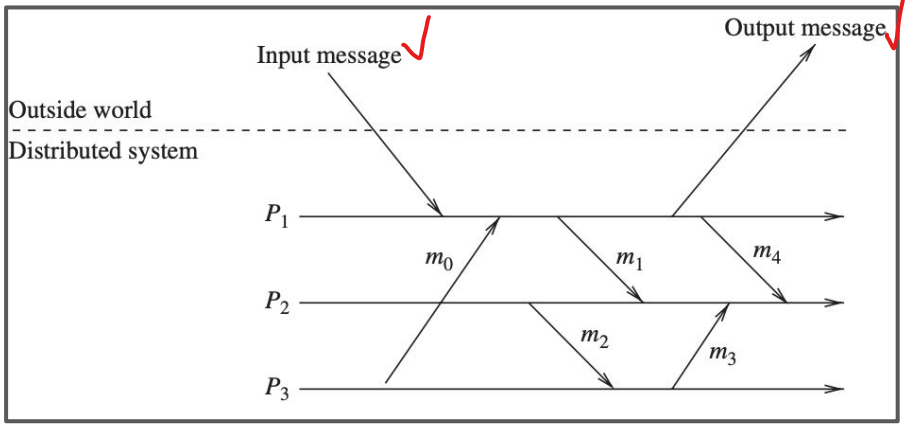
SHARE

SUBSCRIBE

Background and Definitions

1. System Model
2. Local Checkpoint
3. Consistent system states
4. Interactions with the outside world
5. Different types of messages

1. System Model

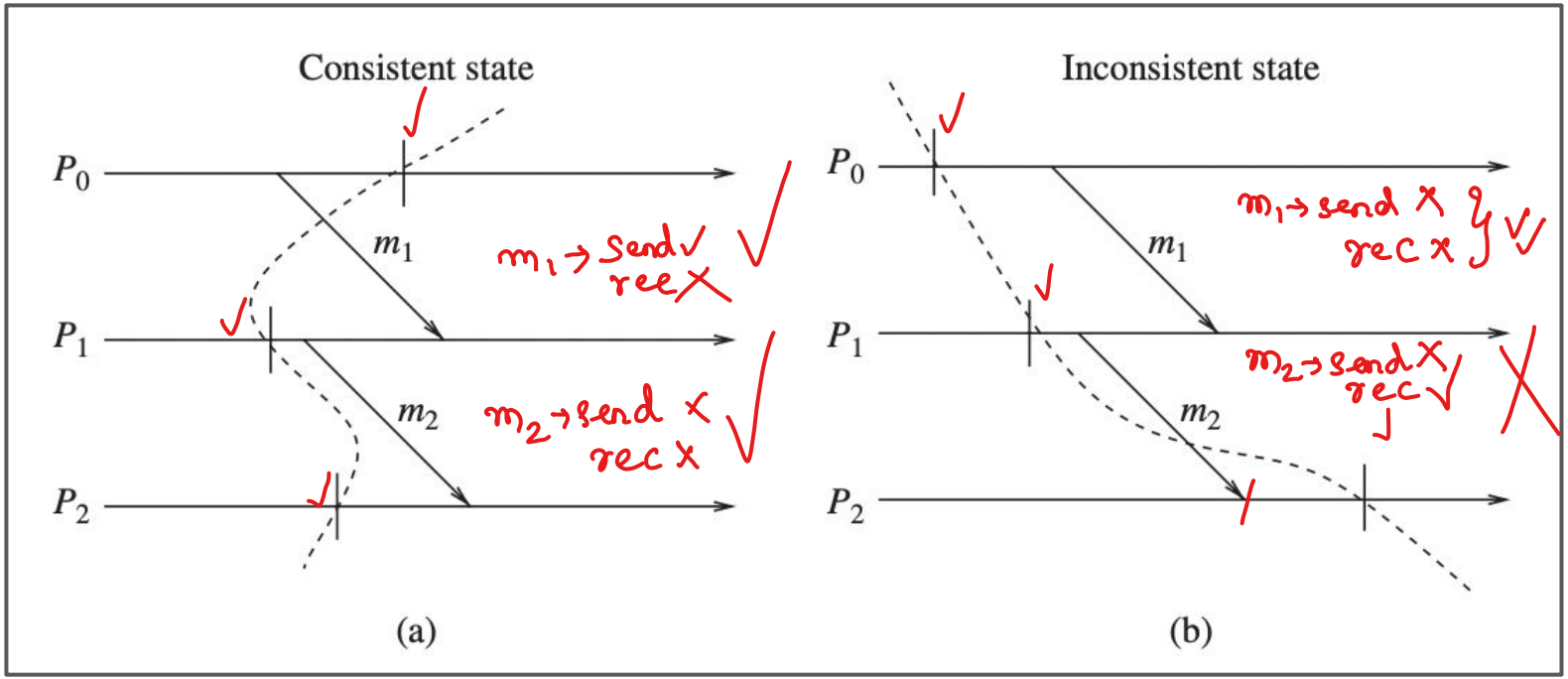


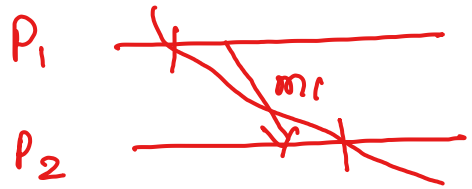
- A distributed system consists of a fixed number of processes, P1, P2 PN , which communicate only through messages.
- Processes cooperate to execute a distributed application and interact with the outside world by receiving and sending input and output messages, respectively.
- Some protocols assume that the communication subsystem delivers messages reliably, in first-in-first-out (FIFO) order, while other protocols assume that the communication subsystem can lose, duplicate, or reorder messages.
- **a system recovers correctly if its internal state is consistent with the observable behavior of the system before the failure**

2. Local Checkpoint - @ each process level

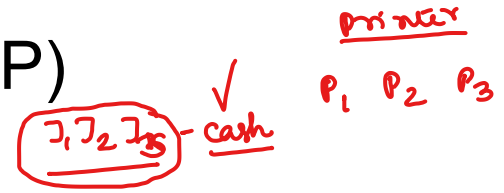
1. A local checkpoint is a snapshot of the **state of the process at a given instance** and the event of recording the state of a process is called local checkpointing.
2. The contents of a checkpoint depend upon the application context and the checkpointing method being used.
3. Depending upon the checkpointing method used, a process may keep **several local checkpoints or just a single checkpoint** at any time
4. a process stores all local checkpoints on the **stable storage** so that they are available even if the process crashes.
5. We also assume that a process is able to **roll back to any of its existing local checkpoints** and thus restore to and restart from the corresponding state

3. Consistent vs Inconsistent System States



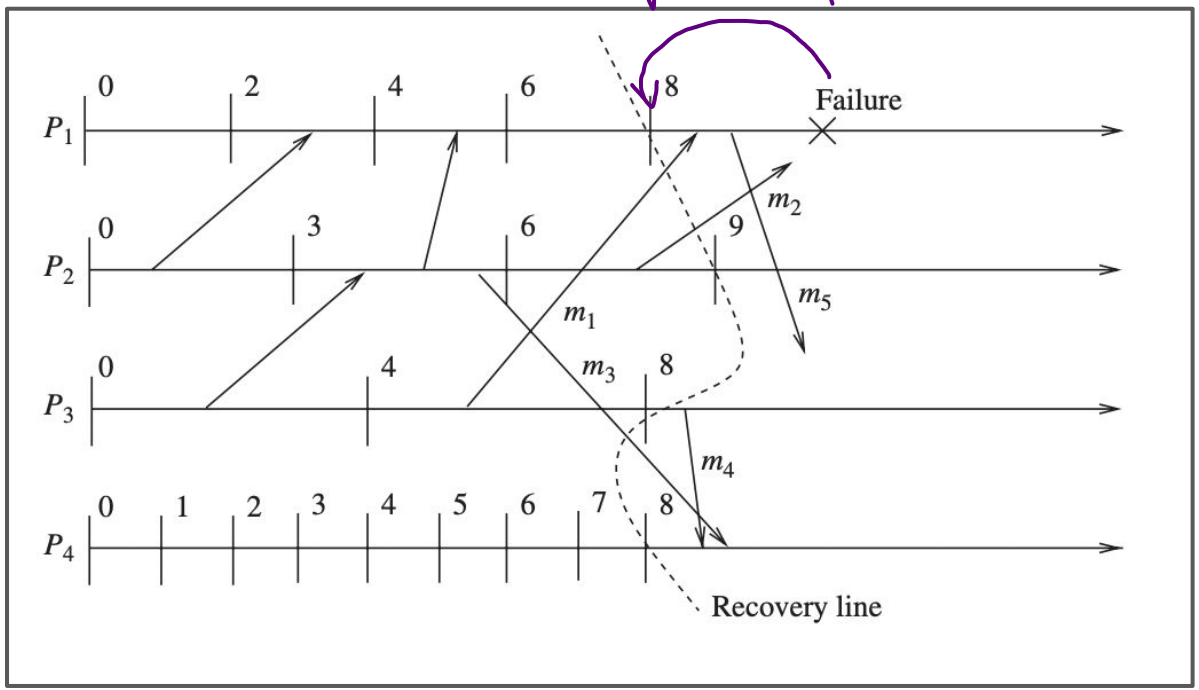


4. Interactions with Outside World (OWP)



- a printer cannot roll back the effects of printing a character, and an automatic teller machine cannot recover the money that it dispensed to a customer
- A distributed application often interacts with the outside world to receive input data or deliver the outcome of a computation. If a failure occurs, the outside world cannot be expected to roll back.
- the outside world see a consistent behavior of the system despite failures
- **Output Commit-** before sending output to the OWP, the system must ensure that the state from which the output is sent will be recovered despite any future failure.
- **Input messages** :
  - Received messages from the OWP may not be reproducible during recovery, because it may not be possible for the outside world to regenerate them.
  - Thus, recovery protocols must arrange to save these input messages so that they can be retrieved when needed for execution replay after a failure

Types of Messages



U selves

① In-transit — Eg. m_2

Sent ✓
receive X

⇒ reliable

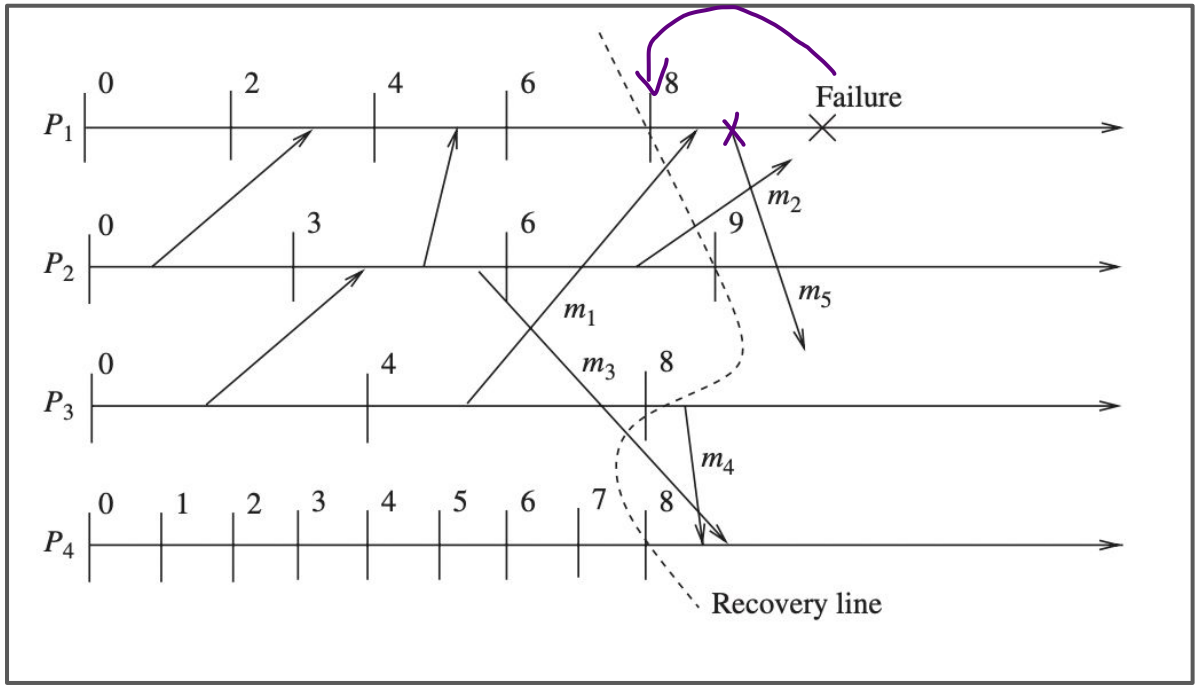
② Lost — Eg. m_1

Sent ✓
receive ⇒ undone

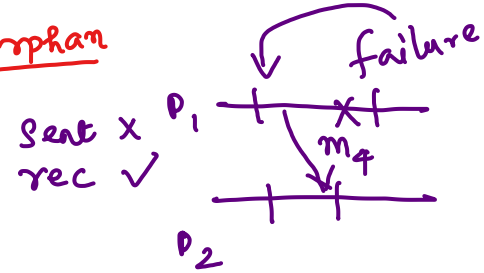
③ Delayed ⓧ Receiver down
ⓧ Receiver rollback

Eg. m_2, m_5

Types of Messages



④ Orphan



⑤ Duplicate

Failure → Rollback → Repeat

Key Points

1. In-transit (m_1,m_2)
   a. Messages that has been sent but not yet received
   b. When in-transit messages are part of a global system state, these messages do not cause any inconsistency.
   c. For reliable communication channels, a consistent state must include in-transit messages because they will always be delivered to their destinations in any legal execution of the system.
   d. On the other hand, if a system model assumes lossy communication channels, then in-transit messages can be omitted from system state.
2. Lost Messages(m_1)
   a. Messages whose send is not undone but receive is undone due to rollback are called lost messages.
   b. This type of messages occurs when the process rolls back to a checkpoint prior to reception of the message while the sender does not rollback beyond the send operation of the message

Key Points….

3. Delayed Messages (m_2,m_5)
   a. Messages whose receive is not recorded because the receiving process was either down or the message arrived after the rollback of the receiving process
4. Orphan Messages
   a. Messages with receive recorded but message send not recorded are called orphan messages.
   b. For example, a rollback might have undone the send of such messages, leaving the receive event intact at the receiving process.
   c. Orphan messages do not arise if processes roll back to a consistent global state.
5. Duplicate Message(m_4,m_5)
   a. Duplicate messages arise due to message logging and replaying during process recovery

:. However, due to the rollback of process P_4 to $C_{4,8}$ and process P_3 to $C_{3,8}$, both send and receipt of message m_4 are undone. When process P_3 restarts from $C_{3,8}$, it will resend message m_4. Therefore, P_4 should not replay message m_4 from its log. If P_4 replays message m_4, then message m_4 is called a *duplicate* message.

Message m_5 is an excellent example of a duplicate message. No matter what, the receiver of m_5 will receive a duplicate m_5 message.

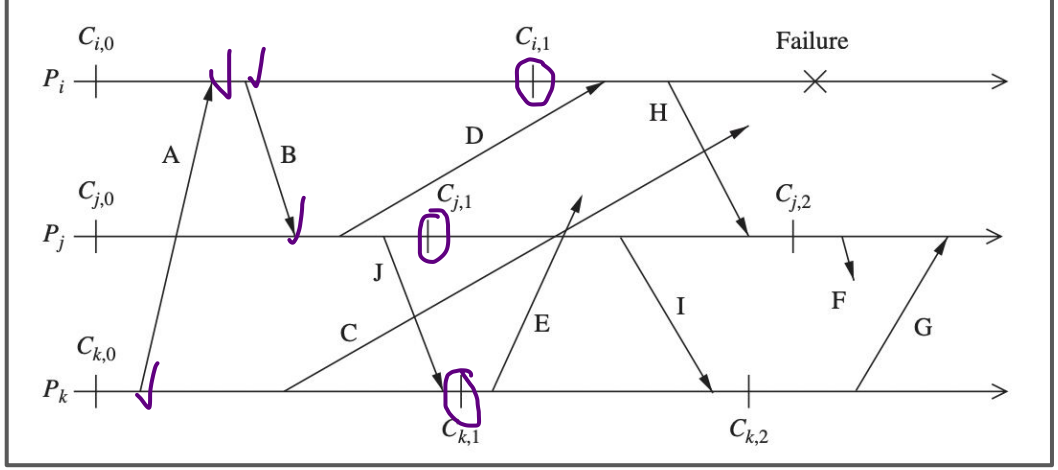
LIKE

COMMENT

SHARE

SUBSCRIBE

Issues in Failure Recovery



A	send✓ rec✓ } no problem
B	
C	→ in-transit (after / before / during)
D	send✓ receive - undone } lost
E	send✓ (undone) receive(?)✓ (orphan → discard)
F	
G	send x receive x } undone (vanished)
H	
I	
J	Send✓ rec✓ no problem

Overlapping failure

Key Points

Messages A, B, D, G, H, I, and J had been received at the points indicated in the figure and messages C, E, and F were in transit when the failure occurred. Restoration of system state to checkpoints $\{C_{i,1}, C_{j,1}, C_{k,1}\}$ automatically handles messages A, B, and J because the send and receive events of messages A, B, and J have been recorded, and both the events for G, H, and I have been completely undone. These messages cause no problem and we call messages A, B, and J normal messages and messages G, H, and I vanished messages [33].

Messages C, D, E, and F are potentially problematic. Message C is in transit during the failure and it is a delayed message. The delayed message C has several possibilities: C might arrive at process P_i before it recovers, it might arrive while P_i is recovering, or it might arrive after P_i has completed recovery. Each of these cases must be dealt with correctly.

Message D is a lost message since the send event for D is recorded in the restored state for process P_j, but the receive event has been undone at process P_i. Process P_j will not resend D without an additional mechanism, since the send D at P_j occurred before the checkpoint and the communication system successfully delivered D.

Messages E and F are delayed orphan messages and pose perhaps the most serious problem of all the messages. When messages E and F arrive at their respective destinations, they must be discarded since their send events have been undone. Processes, after resuming execution from their checkpoints, will generate both of these messages, and recovery techniques must be able to distinguish between messages like C and those like E and F.

LIKE

COMMENT

SHARE

SUBSCRIBE