CSC2/458 Parallel and Distributed Systems
Checkpointing and Recovery

Sreepathi Pai

April 17, 2018

URCS

Outline

Checkpointing and Recovery

Independent Checkpointing

Coordinated Checkpointing

Message Logging

Outline

Checkpointing and Recovery

Independent Checkpointing

Coordinated Checkpointing

Message Logging

Errors happen

- Errors happen
- How do we recover from them (say, for message loss)?
  - (before information theory): ?
  - (after information theory): ?

Checkpointing and Recovery

To checkpoint is to save the state of a computation so that you can "rollback" to it

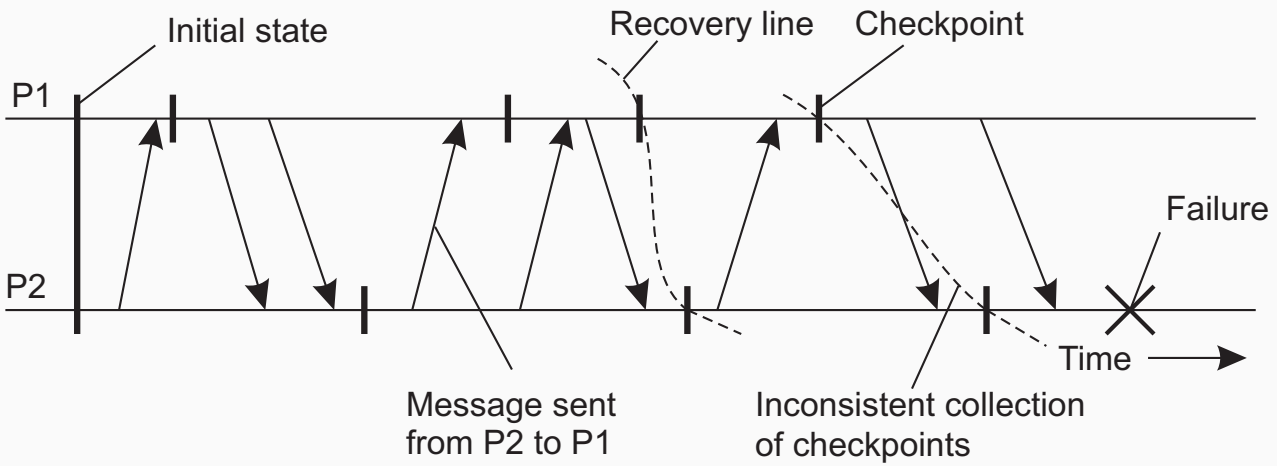
- Examples:
  - Save games
  - Virtual machine snapshots

Recovery is then "simply" restoring the checkpoint

Distributed Checkpointing: The Challenge

- Processes only know:
  - which messages they have received
  - which messages they have sent
  - what their local state is
- Checkpointing ideally should not require everybody to "pause"
  - Must run concurrently with computation

The Recovery Line



Outline

Checkpointing and Recovery

Independent Checkpointing

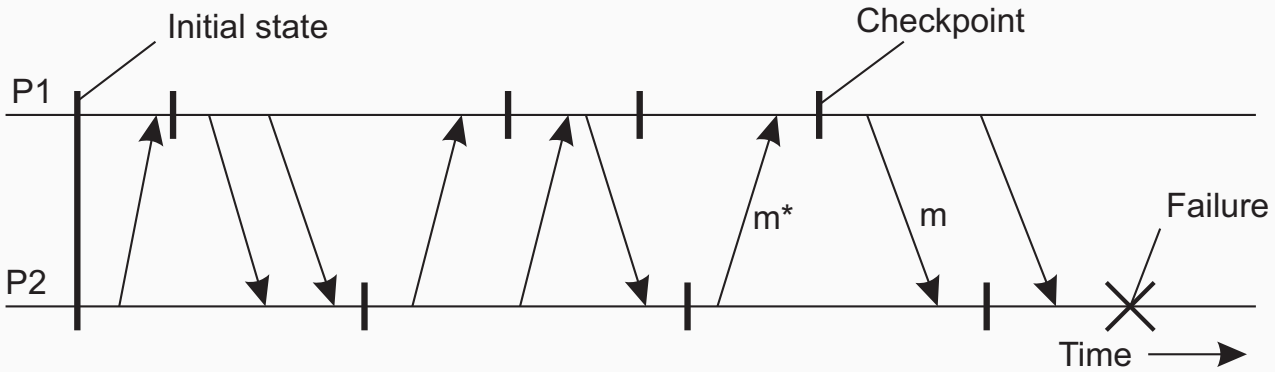
Coordinated Checkpointing

Message Logging

Algorithm

- A process records its local state independently
  - messages sent/received included
- A recovery for a process entails going back to its most recent checkpoint
  - Unfortunately, this can't be done independently

Rollbacks



Assume $P2$ fails. How far we do need to rollback to achieve a consistent worldview?

Detecting dependencies

- For a process P_i, let $INT_i(m)$ be the interval between the $m-1$ and m checkpoints.
- All messages sent in $INT_i(m)$ contain (i, m)
- When process P_j receives this message, it may be in $INT_j(n)$
  - records dependency $INT_i(m) \rightarrow INT_j(n)$
  - saves dependency with checkpoint

Rolling back: Consistency

- If P_i rolls back to checkpoint $m-1$, no messages from $INT_i(m)$ were ever sent
- All checkpoints dependent on $INT_i(m)$ are invalid
- Rollbacks need to continue until consistency is reached

Outline

Checkpointing and Recovery

Independent Checkpointing

Coordinated Checkpointing

Message Logging

Algorithm

- Coordinator broadcasts CHECKPOINT-REQUEST message to all processes
- When this request is received,
  - Process checkpoints local state
  - Acknowledges to coordinator that it has taken checkpoint and waits
- When coordinator receives acknowledgements from all processes, it sends CHECKPOINT-DONE
  - Processes resume computation
- What about messages?

Message handling

- All incoming messages received after CHECKPOINT-REQUEST are not considered part of the checkpoint
- All outgoing messages are held back until CHECKPOINT-DONE is received
- This results in a "globally consistent state"
  - How?

Outline

Checkpointing and Recovery

Independent Checkpointing

Coordinated Checkpointing

Message Logging

Basic idea

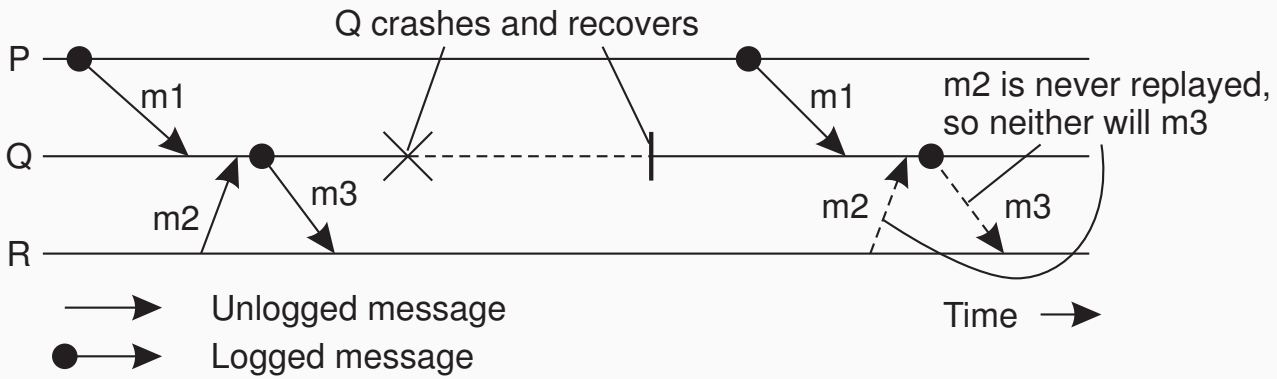
- Computations are deterministic and rely only on messages transmitted
- Save messages from a checkpoint and replay them during recovery

Piecewise deterministic execution

- A piecewise deterministic computation interval:
  - starts with a non-deterministic event (e.g. receipt of a message)
  - continues in a completely deterministic fashion
  - ends just before another non-deterministic event

This implies that only non-deterministic events need to be logged.

Who should save the messages?



Orphan processes

- Let $DEP(m)$ represent processes that depend on message m
- Let $COPY(m)$ represent processes that contain a copy of m
  - but may not have logged it
- Note, m contains all details necessary to retransmit it

A process Q is orphaned if and only if:

- Q depends on m (i.e. $Q \in DEP(m)$)
- All processes in $COPY(m)$ have failed
- So m cannot be played back

Pessimistically avoiding orphan processes

- Orphan processes can be avoided by ensuring that
  - A non-deterministic message is sent only to one process
  - That process cannot send another message without logging m

Further reading

Chandy and Lamport, “Distributed Snapshots: Determining Global States of Distributed Systems”, ACM TOCS 1985

Acknowledgments

All figures from Van Steen and Tanenbaum, Distributed Systems, 3rd Edition, Chapter 8.