

## Fault Tolerance and Recovery in Distributed Systems

70

## Fault Tolerance

- Fault – cause of an error that might lead to failure; could be transient, intermittent, or permanent
- Fault tolerance – a system can provide its services even in the presence of faults
- Requirements
  - Availability – most likely working at any instant
  - Reliability – time interval between failures
  - Safety – no catastrophe in operation
  - Maintainability – ease of repair

71

## Failure Models

- Crash/fail-stop: correct operation until system is unresponsive/halts
  - Fail-silent
  - Fail-safe
- Omission: request or response dropped
- Timing: outside specified real-time requirements
- Response failure: incorrect data or state transition
- Byzantine or arbitrary: malicious or incorrect

72

## Failure Masking by Redundancy

- Information, e.g., ECC
- Time, e.g., retransmission or repeat
- Physical, e.g., triple modular redundancy

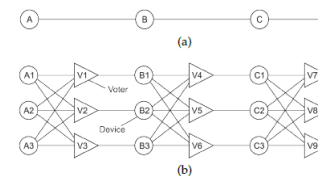


Figure 8.3: Triple modular redundancy.

From van Steen and Tanenbaum

73

## Checkpointing, Logging, and Rollback Recovery

- Approaches to reliability
  - Transactions: data-oriented applications
  - Group communication: abstraction of ideal communication system
  - Rollback recovery: long-running applications

A system recovers correctly if its internal state is consistent with the observable behavior of the system before the failure

74

## Background

- Message-passing system with N processes
  - Possible to model shared memory using message passing
- Fail-stop failures possible at any time
- GOAL: recover all processes to some “consistent” state after one or more nodes have failed and then recovered (or been replaced)

75

## Background and Definitions

- System model
  - Collection of application processes, communicating thru a network
  - Processes have access to a *stable storage* device, where recovery info is periodically saved during failure-free execution
  - Recovery info includes:
    - Checkpoints
    - logs of interactions with I/O devices
    - events that occur at each process
    - messages exchanged among processes
  - Protocols may assume the communication subsystem is reliable & FIFO, or unreliable (lost, duplicate, reordered messages)

76

## Checkpointing

- Basic idea: If something goes wrong, we just pick up the pieces and put everything back where it was
- At regular intervals, dump all memory and resources associated with a process to persistent storage
- When a process fails, we reload the stored file into memory, restore all state, and restart the process from where it was when the checkpoint was taken

77

## Concurrent Checkpointing

- Idea: trap writes to pages you haven't checkpointed, and copy the original pages to a buffer before allowing the write
- In filesystems, this is called “snapshotting”
- Advantage: the process may continue computing while the checkpointing takes place

78

## Incremental Checkpointing

- Idea: save only MODIFIED pages
- Best when used together with concurrent checkpointing
- Could even use differential storage to reduce space overhead of multiple checkpoints for a single process

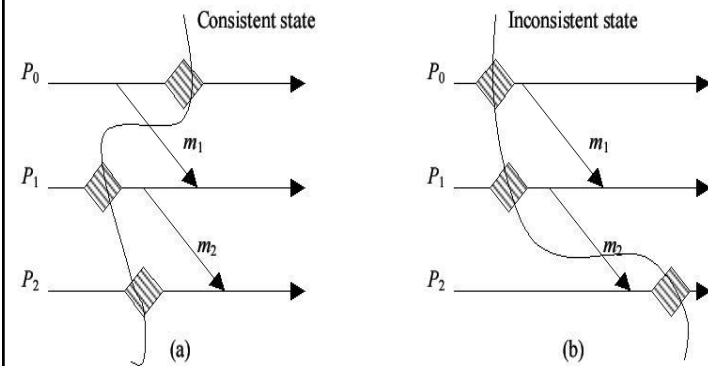
79

## Checkpointing in a Distributed Environment

- Consistency Issue: if P1 checkpoints and then sends a message to P2, you can no longer roll P1 back to before the message was sent unless P2 rolls back to before the message's receipt
- Issue: Checkpointing can be very expensive; you don't want to checkpoint more than necessary, but you also don't want to have to roll back too far
- Issue: Communication with the outside world doesn't get undone

80

## Illustration of Consistency Problems



81

# Uncoordinated Checkpointing

- Every once in a while, a process will checkpoint its own state
- For recovery, each process sends information on the checkpoints it has to some central point, which calculates the “recovery line” - the last set of checkpoints to create a globally consistent state

82

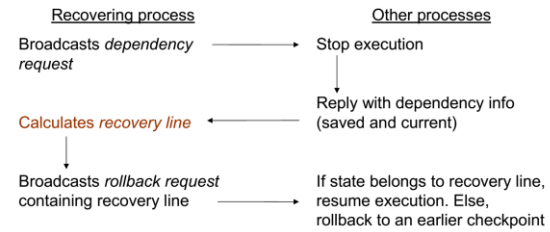
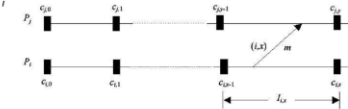
## 1. Uncoordinated checkpointing

- Each process independently decides when to take checkpoints

-  $c_{i,x}$ :  $x^{\text{th}}$  checkpoint of process  $P_i$

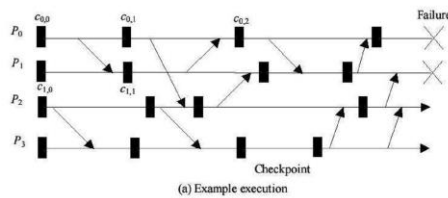
-  $I_{j,y}$ : checkpoint interval

Maximum number of useful checkpoints that must be kept on stable storage cannot exceed  $N(N+1) / 2$



83

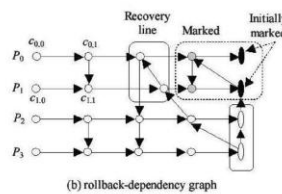
## 1. Uncoordinated checkpointing



### Rollback dependency graph

Directed edge drawn from  $c_{i,x}$  to  $c_{j,y}$  if either

- (1)  $i \neq j$ , and a message  $m$  is sent from  $I_{i,x}$  and received in  $I_{j,y}$ , or
- (2)  $i = j$  and  $y = x + 1$



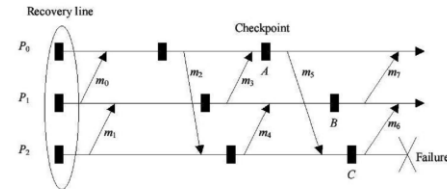
84

## 1. Uncoordinated checkpointing

Advantage: each process may take checkpoint when most convenient

### Disadvantages

- Domino effect



- Useless checkpoints

- Each process must maintain multiple checkpoints
- Garbage collector must be invoked periodically
- Not suitable for apps with frequent output commits

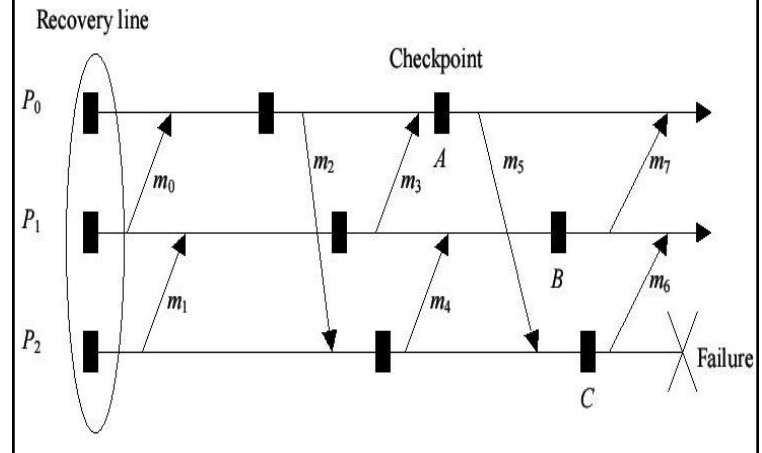
85

## Problems with Uncoordinated Checkpointing

- The “domino effect”
- Garbage collection of old checkpoints
- Many useless checkpoints
- Communication with OWP requires global information

86

## Domino Effect



87

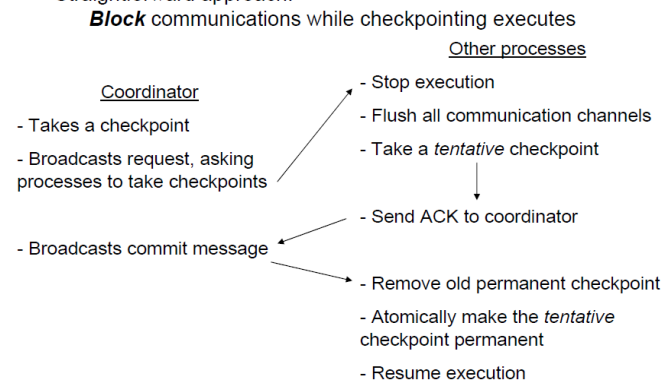
## Coordinated Checkpointing

- Idea: all message-passing calls block while checkpointing takes place at the same time across all processes
- Checkpoints are only valid after it is known that all processes have successfully completed their own copy
- Advantages:
  - only one checkpoint needs to be kept
  - recovery is just rolling back everything to THE checkpoint
- Disadvantage: overhead due to blocking

88

## 2. Coordinated checkpointing

Straightforward approach:



89

## Nonblocking Checkpoint Coordination [e.g., Chandy&Lamport 1985 Distributed Snapshots]

- FIFO channels:
  - Precede 1st post-checkpoint message on each channel by a checkpoint request
  - Force each process to take a checkpoint upon receiving 1st checkpoint request
- Non-FIFO channels:
  - Checkpoint request piggybacked on every post-checkpoint message

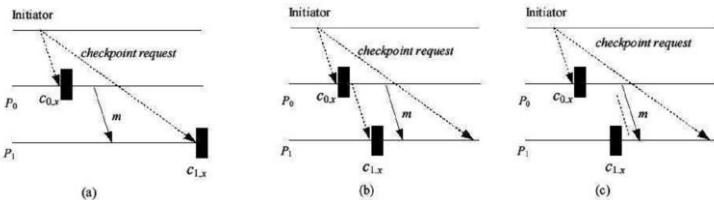


Figure 8. Non-blocking coordinated checkpointing: (a) checkpoint inconsistency; (b) with FIFO channels; (c) non-FIFO channels (short dashed line represents piggybacked checkpoint request).

90

## Coordinated Checkpointing

- Advantages
  - Simplified recovery
  - Avoids domino effect
  - Less stable storage requirement (1 checkpoint / process)
  - No need for garbage collection
- Disadvantages
  - Large latency in committing output (may improve by minimal checkpoint coordination)

91

## Logging as a Complement to Checkpointing

- The Piecewise Deterministic (PWD) Assumption
  - Any nondeterministic element of the system may be captured in such a way as to allow replaying it in a deterministic fashion at a later time
- Determinant – Information necessary to replay a nondeterministic event during recovery, logged during failure-free operation
- OWP – Outside World Process: A “process” whose incoming messages represent program output and whose outgoing messages are program input
- “In Transit” - a message sent but not received
- “Orphan” - a process whose checkpoint depends on a state that cannot be recreated

92

## Pessimistic Logging

- Write determinant to stable storage before allowing further events
  - observable pre-failure state always recoverable
  - Recovery always starts from the most recent checkpoint
  - Effect of failure confined to the processes that fail
  - Simple garbage collection: reclaim everything before last checkpoint

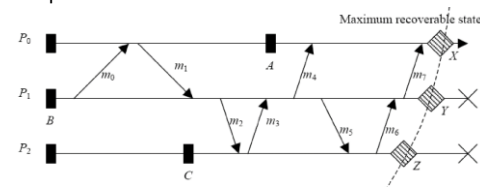


Figure 10. Pessimistic logging.

93