Recovery

CS239
Advanced Operating Systems

November 13, 1997
Dr. Richard Guy

Lecture Source Materials

Singhal, Ch. 12: **Recovery**

Outline

- Introduction
- Concepts
- Recovery in concurrent systems
- Consistent checkpointing
- Synchronous checkpointing and recovery
- Asynchronous checkpointing and recovery
- Summary

Introduction

- recovery refers to restoring a (damaged) system to its normal operational state
  – restart process/processor
  – reclaime resources
  – undo/finish incomplete “transactions”
  – concurrency makes things even harder

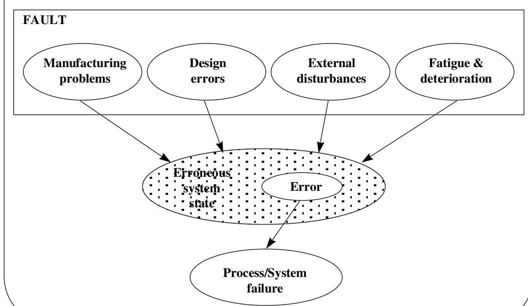
Concepts

- Definition of failure
- Classification of failures
- Forward error recovery
- Backward error recovery

Definition of failure

- “Failure of a system occurs when the system does not perform its services in the manner specified.”
- Fault is anomalous physical condition
  – includes system specification/implementation mistakes
  – results in erroneous system state (→ failure)
    - error is part of system state which differs from its intended value

Failure sources



Classification of failures

- Process failure
- System failure
- Secondary storage failure
- Communication medium failure

Process Failure

- Examples
  - – Computation results in incorrect outcome
  - – System state deviates from specification
  - – Process fails to progress
- Errors leading to failure
  - – deadlock, timeout, protection violation
  - – bad input, consistency violation (optimistic CC)
- Ignoring malicious behavior

System Failure

- Processor fails to execute
  - – software error, hardware error (CPU, bus, etc.)
- Fail-stop behavior assumed
- Failure types
  - – amnesia
  - – partial-amnesia
  - – pause
  - – halting

Secondary Storage Failure

- Stored data inaccessible
  - – parity error
  - – head crash
  - – contaminated medium
- Reconstructable from archive + log, maybe
- Mirrored disks (+independent failure mode)

Comm. Medium Failure

- Site can't communicate with another site
- Causes
  - – switching node failure
    - hardware failure
    - software failure (Internet collapse, Oct. 1980)
      - – Internet RFC 789; ACM Soft. Eng. Notes, Jan. 81.
      - – www.mathcs.carleton.edu/students/johnsond/project.html
    - congestion

Communication failures, cont.

- Link failure
  - hardware
- Implementation failure
  - Internet partition of northeast US
    - multiple “independent” logical circuits on one fiber
- Network partitions can result

Forward error recovery

- Goal: to restore system from erroneous state to error-free state
- If nature of error is completely known
  - remove error from state
  - proceed with execution from error-free state
- Rarely possible to do...

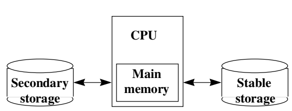
Backward error recovery

- when error sources unknown
  - restore state to previous error-free state; restart
  - independent of fault, errors causing fault
  - problems
    - performance penalty
    - no guarantee fault will not reoccur
    - possible unrecoverable component of state
- recovery point: state used to replace error

Backward error recovery, cont.

- basic approaches
  - model
  - operation-based
  - state-based

Backward error recov. model



User data

Logs
Recovery points

Operation-based recovery

- Basic idea
  - audit trail (log) has details for reversing all state changes
- Update-in-place
- Write-ahead-log

Update-in-place

- every update to object also records in log
  - name of object
  - old state of object (used for UNDO)
  - new state of object (used for REDO)
- recoverable update operation implements as
  - *do* operation
  - *undo* operation (given *do* log, undoes *do* operation)
  - *redo* operation (given *do* log, redoes *do* operation)
- undo partial update; redo from archive

Write-ahead-log

- update-in-place has problem if crash occurs between update & log recorded to stable sto.
- update object only after undo log recorded
- before commiting updates, record both redo and undo logs
- expensive to write log to stable storage...

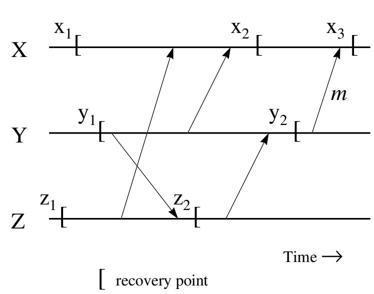
State-based recovery

- Save entire process state at recovery point
  - checkpointing
  - recovery point called *checkpoint*
  - *rolling back* process: restoring to checkpoint
  - tradeoff: frequent CPs vs. completion delay
- Shadow pages
  - save unmodified page copy on stable storage
  - update only volatile copy; discard on roll back

Concurrent Systems Recovery

- Rolling back failed process may require roll back of active process
  - need cooperative recovery point determination
- Roll back issues
  - Orphan messages & domino effect
  - Lost messages
  - Livelocks

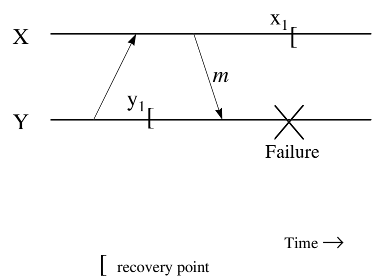
Orphan messages



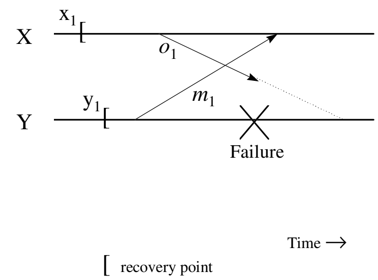
Domino effect

- Suppose Y rolls back to checkpoint y_2
  - *m* is orphan message
  - process X must rollback (before x_3) to x_2
- Suppose Z rolls back to checkpoint z_2
  - Y rolls back to y_1
  - forcing Z to roll back to z_1 (!!!)

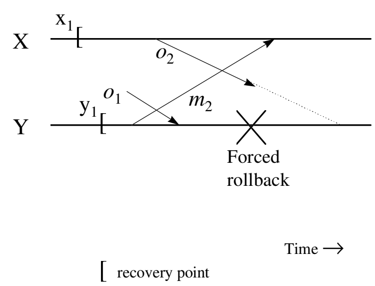
Lost messages



Livelocks (part 1)



Livelocks (part 2)



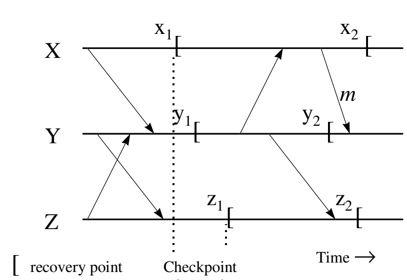
Concurrent recovery...

- Operation- or state-based recovery are NOT adequate for locating/establishing recovery points for cooperating processes.
- Coordination required at either
  – time of establishing checkpoints
  – beginning of recovery

Consistent Checkpointing

- Local checkpoint
- Global checkpoint: set of local checkpoints
- Strongly consistent checkpoint set
  – no information flow between set of processes during checkpoint interval (earliest to latest)
- Consistent checkpoint set
- Simple consistent checkpoint method

Consistent checkpoint set



Synchronous Checkpointing

- Checkpoint + recovery = solution
- Checkpoint algorithm (Koo & Toueg)
- Recovery algorithm (Koo & Toueg)

Checkpoint alg assumptions

- communication via messages on channels
- unreliable FIFO channels
  - higher level end-to-end protocols assumed
  - subsumes rollback-caused message loss
- no network partitions from comm. failures
- CP alg invoked by single process
- no site failures during CP execution

Checkpoint alg concepts

- Permanent and tentative checkpoints
  - saved on stable storage
  - permanent: part of known consistent global CP
  - tentative: until succesful termination of CP alg.
  - rolls back only to permanent checkpoint
- No messages sent between tentative CP and before decision received

Two-phase synch CP algorithm

- Phase one
  - P takes tentative checkpoint
  - request all other processes to take tentative CP
  - other processes reply with success/failure
  - if all reply “success”, P decides all tentative CPs should be made permanent; otherwise, P decides all tentative CPs should be discarded

Two-phase CP algorithm...

- Phase two
  - P tells all processes of decision from phase one
  - all processes act accordingly
- Therefore, all/none take permanent CP
- Correctness
  - all/none take permanent checkpoints
  - no inconsistencies
    - no messages recorded received, but not sent

Rollback recovery algorithm

- Assumptions
  - single process invocation
  - not concurrently invoked with checkpoint
  - messages blocked during RR execution
- Two phase algorithm (again)

Two-phase RR algorithm

- Phase one
  - – P inquires all processes if willing to restart from (permanent) checkpoint
  - – all processes respond yes/no
  - – if all say yes, P decides all should restart; otherwise, P decides continue (may try later)
- Phase two
  - – P tells all processes of decision; do as told
- Correctness: all/none; CP is consistent

Asynchronous Checkpointing

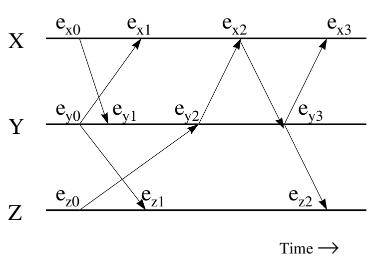
- Problems with synchronous
  - – Additional messages sent by CP algorithm
  - – Synchronization delays introduced (no msgs)
  - – If failures rare between CPs, above costs a lot
- Asynchronous
  - – local checkpoints taken independently
  - – but, no guarantee a set of CPs is consistent
  - – recovery alg must search for consistent set

- log all incoming messages on stable storage
  - minimizes undone computation
  - allows reprocessing of messages after roll back
  - pessimistic logging
    - – log before processing by computation
    - – slows down computation, even if failure-free
  - optimistic logging
    - – concurrently save messages in volatile storage
    - – log to stable storage at intervals
    - – trade off redone computation vs slow down

Asynchronous CP/R algorithm

- Assumptions
  - – reliable FIFO communications channels
  - – infinite channel buffers/finite message tx delay
  - – event-driven computation
    - P idle until message received
    - processes message and changes state
    - sends zero or more messages
    - can ID each event with monotonically incr. counter

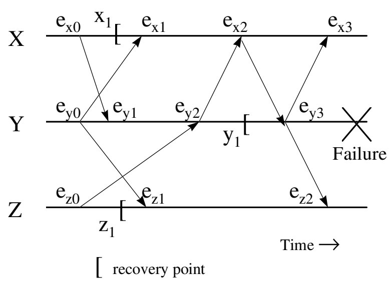
Event driven computation



Asynchronous CP, cont.

- Basic idea
  - – save {s, m, msgs_sent} at each event
  - – volatile log (fast) flushed to stable log (slower)
  - – each processor notes number of messages sent to others, and received from others
  - – use counters to determine orphan messages
  - – want to find state where received= =sent to it

Event driven rollback



Summary

- Failures caused by errors
- Can remove errors by forward/backward ER
- Backward ER more costly, more general
- Synchronous checkpoints helpful, costly
- Asynchronous CPs messier, domino effect