CS 138: Ordering and Global State
Administrivia

• HW2 is out today, due on the 15th (1 week)

• Review session will be on Monday, March 21st, 5:30pm

• Midterm will be on Tuesday, March 22nd, with material up to Raft (next two classes)
Global State
Failure Happens

• What to do about it?
  – you of course have everything backed up
  – so, restore the backups
Global State

• Your system consists of 100 nodes
  – each produces a snapshot of itself periodically
  – does some collection of these snapshots constitute a meaningful notion of “global state”?
Distributed Snapshots (1)

- \( m_1 \)
- \( m_2 \)
- \( m_3 \)
- \( m_4 \)
- \( m_5 \)
- \( m_6 \)
A cut is a **consistent cut** if, for each event $e$ it contains, it also contains all events that happened before $e$. 
Checkpointing

• Produce a distributed snapshot
  – how?

• Independent checkpointing
  – each process checkpoints itself periodically when convenient
  – to produce distributed snapshot
    - start with most recent checkpoints
    - roll back until consistent global checkpoint is achieved
Independent Checkpointing

A

B

C

m1

m2

m3

m4

m5

m6

Roll back
Domino Effect

P1

Initial state

Checkpoint

P2

Failure

Time
Coping

• Take independent, periodic checkpoints, plus a few more

or

• Produce a global snapshot on demand
Independent Checkpoints

• Goal
  – all checkpoints are “useful”
    - no need to roll back
• What are the conditions for checkpoints to for a consistent cut?
Causal Paths

P_1

C_{1,0} \quad C_{1,1} \quad C_{1,2}

m_1 \quad \text{checkpoint interval} \quad m_3

P_2

C_{2,0} \quad C_{2,1} \quad C_{2,2}

m_2 \quad m_4

P_3

C_{3,0} \quad C_{3,1} \quad C_{3,2}
Causal Paths

P1

C1,0

C1,1

checkpoint interval

C1,2

m1

P2

C2,0

C2,1

m2

C2,2

m3

P3

C3,0

C3,1

m4

C3,2
Non-Causal Paths

\[ P_1 \quad C_{1,0} \quad C_{1,1} \quad \text{checkpoint interval} \quad C_{1,2} \quad P_2 \]
\[ P_3 \quad C_{3,0} \quad C_{3,1} \quad C_{3,2} \]

\[ m1 \quad m2 \quad m3 \quad m4 \]
Zigzag Paths

$C_{1,0}$ $C_{1,1}$ checkpoint interval $C_{1,2}$

$m1$ $m3$

$C_{2,0}$ $C_{2,1}$

$m2$ $m4$

$C_{3,0}$ $C_{3,1}$

$C_{3,2}$
Zigzag Path Definition

• A zigzag path exists from $C_{p,i}$ to $C_{q,k}$ iff there are messages $m_1, m_2, \ldots, m_n$ such that
  - $m_1$ is sent by process $p$ after $C_{p,i}$
  - if $m_h$ ($1 \leq h \leq n$) is received by process $r$, then $m_{h+1}$ is sent by $r$ in the same or a later checkpoint interval (although $m_{h+1}$ may be sent before or after $m_h$ is received), and
  - $m_n$ is received by process $q$ before $C_{q,k}$
Theorem

• A set of checkpoints S, each from a different process, can belong to the same consistent global snapshot iff no checkpoint in S has a zigzag path to any other checkpoint (including itself) in S
Zigzag Cycles

$P_1 \quad C_{1,0} \quad C_{1,1} \quad C_{1,2} \quad P_2 \quad C_{2,0} \quad C_{2,1} \quad C_{2,2} \quad P_3 \quad C_{3,0} \quad C_{3,1} \quad C_{3,2}$

$m_1 \quad m_2 \quad m_3 \quad m_4$
Corollary

• A checkpoint is *useful* if it potentially belongs to some consistent global checkpoint
• Corollary: A checkpoint is useful iff it is part of no zigzag cycle
Adaptive Checkpointing

• On receipt of a message, receiver checks if message completes a zigzag cycle
  – if so, a new checkpoint is taken before the message is processed
  – thus, no cycle
However ...
Coping ...

- On receipt of message, check for a causal path to a checkpoint preceding the send
  - the path plus the just-received message form a zigzag cycle
- Doesn’t catch all zigzag cycles
  - testing shows it catches most of them
Finding Causal Paths

• Use vector clocks
  – components are counts of checkpoints in each process
  – details may be an exercise …
Producing a Consistent Global Snapshot on Demand

• Process A wants all other processes to send it snapshots that together form a consistent cut (and thus a global snapshot)
• Can this be done?
Distributed Snapshot Algorithm

- Chandy & Lamport, 1985
  - algorithm to select a *consistent cut*
  - any process may initiate a snapshot at any time
  - processes can continue normal execution
    - send and receive messages
  - assumes:
    - no failures of processes & channels
    - strong connectivity
      - at least one path between each process pair
    - unidirectional, FIFO channels
    - reliable delivery of messages
Approach

• Snapshot consists of saved states of all nodes along with messages in transit

• For each pair of directly connected nodes A and B
  – must record messages sent before A saved its state but received after B saved its state
  – nodes send out special *marker* messages immediately after saving their states
Example: Sending

\[ p_1 \rightarrow m_3 \rightarrow M \rightarrow m_2 \rightarrow m_1 \rightarrow p_2 \]

state

3
Example: Receiving
Another Example: part 1
Another Example: part 2

p_1 \rightarrow \text{state} \rightarrow p_3 \rightarrow \text{state} \rightarrow p_2 \rightarrow \text{state}

m_{12} \rightarrow M \rightarrow 1 \rightarrow M \rightarrow 2

M \rightarrow 3 \rightarrow M \rightarrow 4
Another Example: part 3

Diagram with states and transitions:
- \( p_1 \) to \( p_3 \)
- \( p_3 \) to \( p_2 \)
- \( r(m2) \)

States:
- \( p_1 \)
- \( p_2 \)
- \( p_3 \)
- \( r(m2) \)

Transitions:
- \( m3 \)
- \( m2 \)

Notes:
- M
- 3
- 5
- 6
- 1
- 2

Copyright © 2015 Thomas W. Doeppner, Rodrigo Fonseca. All rights reserved.
Another Example: part 4

p₁ \rightarrow \text{state} \rightarrow p₃ \rightarrow p₂ \rightarrow \text{state r(m2)}

\text{m₃} \rightarrow 2

1
Snapshot Rules

• Marker receiving rule for process $p_i$
  On $p_i$’s receipt of a marker message over channel $c$:
  
  \[\text{if (} p_i \text{ has not yet recorded its state) }\]
  
  it records its state
  it records the state of $c$ as the empty sequence
  it turns on recording of messages arriving over other channels

  \[\text{else}\]
  
  $p_i$ records the state of $c$ as the set of messages it has received over $c$ since it saved its state and before it received the marker over $c$

• Marker sending rule for process $p_i$
  After $p_i$ has recorded its state, for each outgoing channel $c$:
  
  $p_i$ sends one marker message over $c$ (before it sends any other messages over $c$)
Termination

• Process P has completed its part of the algorithm when it has processed markers on all input channels

• It sends its saved local state and channel histories to the initiator
  – the intent is that collection of local states form consistent cut
    - channel histories are the messages in transit at time of cut
Analysis

• Does it find a consistent cut?
  – if so, then for any $P_a$ and $P_b$, if $m$ is a message sent from $P_a$ to $P_b$, then if $\text{recv}(m)$ is in the cut, so is $\text{send}(m)$
    - i.e., if $\text{recv}(m)$ occurred before $P_b$ recorded its state, then $\text{send}(m)$ occurred before $P_a$ recorded its state
  – stronger statement: if for any $P_a$ and $P_b$, if $e_a$ and $e_b$ are events in $P_a$ and $P_b$, such that $e_a$ happens before $e_b$ ($e_a \rightarrow e_b$), then if $e_b$ is in the cut, so is $e_a$
    - i.e., if $e_b$ occurred before $P_b$ recorded its state, then $e_a$ occurred before $P_a$ recorded its state
Proof

• Assume no: $P_a$ recorded its state before $e_a$ occurred ($e_b$ is in the cut, but $e_a$ is not)
  - since $e_a \rightarrow e_b$, there was some sequence of messages $m_1, m_2, \ldots, m_h$ that brought on $e_a \rightarrow e_b$
  - since $P_a$ recorded its state before $e_a$ occurred, it sent marker messages out on all its outgoing channels before transmitting $m_1$
  - since the channels are FIFO, a marker reached $P_b$ before $m_h$
  - but then $P_b$ would have recorded its state before $e_a$
  - but then $e_b$ would not have been in the cut
    - contradiction
More Analysis

• Snapshot taken isn’t necessarily a state that actually happened!
  – but it could have happened …

• If distributed system deadlocks, no distributed snapshot
Example (part 1)

\[ p_1 \xrightarrow{c_2} p_2 \]

\[ p_1 \xleftarrow{c_1} p_2 \]

$1000 \quad (\text{none}) \quad $50 \quad 2000

account \quad \text{widgets} \quad \text{account} \quad \text{widgets}
Example (part 2)

1. Global state $S_0$
   
   \[
   \langle 1000, 0 \rangle \xrightarrow{c_2} \langle 50, 2000 \rangle
   \]

2. Global state $S_1$
   
   \[
   \langle 900, 0 \rangle \xrightarrow{c_2} \langle 50, 2000 \rangle
   \]

3. Global state $S_2$
   
   \[
   \langle 900, 0 \rangle \xrightarrow{c_2} \langle 50, 1995 \rangle
   \]

4. Global state $S_3$
   
   \[
   \langle 900, 5 \rangle \xrightarrow{c_2} \langle 50, 1995 \rangle
   \]

(M = marker message)
Reachability

actual execution: \( e_0, e_1, \ldots, e_N \)

pre-snap: \( e'_0, e'_1, \ldots, e'_{R-1} \)

post-snap: \( e'_R, e'_{R+1}, \ldots, e'_N \)
Global Properties

• Safety
  – bad things will not happen
  – e.g., mutual exclusion is a safety property

• Liveness
  – good things will happen
  – e.g., termination is a liveness property

• Stable properties
  – once true — always true

• Transient properties
  – once true — who knows?
Stable Global Properties

(a) Garbage collection

(b) Deadlock

(c) Termination
Transient Properties

• Distributed debugging
  – assert(∀a≠b (|x_a - y_b| < 10))
    - x_a and y_a reside in process a
How To ...

• State collection
  – each process sends snapshots to central server
  – contain vector timestamps

• Central server checks for transient property $\phi$
  – looks at global states that could have resulted from initial state, given vector timestamps

• possibly $\phi$
  – if $\phi$ holds in at least one of them

• definitely $\phi$
  – for all possible (causally consistent) orderings, $\phi$ holds at some point