CSE 486/586 Distributed Systems
Global States

Steve Ko
Computer Sciences and Engineering
University at Buffalo

Last Time

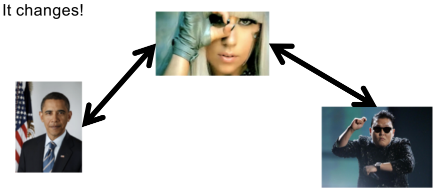
- Ordering of events
  - Many applications need it, e.g., collaborative editing, distributed storage, etc.
- Logical time
  - Lamport clock: single counter
  - Vector clock: one counter per process
  - Happens-before relation shows causality of events

Today's Topic

- Global snapshots
- An "application" of logical time
- Today's topic will deepen your understanding about causality and the abstract view of distributed systems.

Today's Question

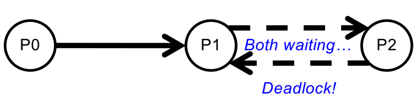
- Example question: who has the most friends on Facebook?
- Challenges to answering this question?
  - It changes!



- What do we need?
  - A snapshot of the social network graph at a particular time

Today's Question

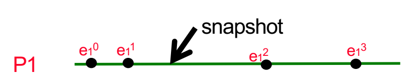
- Distributed debugging



- How do you debug this?
  - Log in to one machine and see what happens
  - Collect logs and see what happens
  - Taking a global snapshot!

What is a Snapshot?

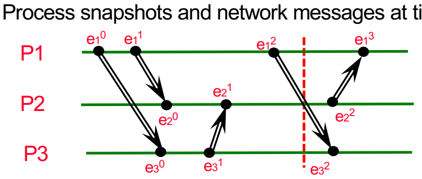
- Single process snapshot
  - Just a snapshot of the local state, e.g., memory dump, stack trace, etc.
  - For the sake of this lecture, let's say a log of events.
  - When we capture a snapshot, we want to be able to trace the causality (e.g., important for debugging).



- Let's say we're logging all events.
  - The above snapshot (a dump of log messages) will include e_1^0 and e_1^1. This allows us to trace the causality of events.
  - How to do this for a multiple processes?

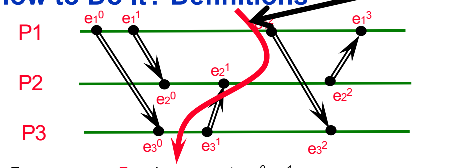
Ideal: Instantaneous Snapshot

- Process snapshots and network messages at time t



- The most general multi-process snapshot that can explain all causality
  - Causality across processes
  - Messages caused by send events
- But we can't quite do it due to imperfect clock sync.
- We do it thru logical snapshots.

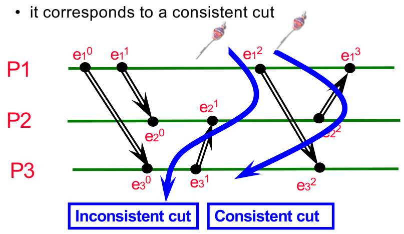
How to Do It? Definitions



- For a process P_i, where events e_i^0, e_i^1, … occur,
  - history(P_i) = h_i = <e_i^0, e_i^1, … >
  - prefix history(P_i^k) = h_i^k = <e_i^0, e_i^1, …,e_i^k >
  - S_i^k : P_i's state immediately after k^th event
- For a set of processes P_1, …,P_i, …. :
  - Global history: $H = \cup_i (h_i)$
  - Global state: $S = \cup_i (S_i^{k_i})$
  - A cut $C \subseteq H = h_1^{c1} \cup h_2^{c2} \cup \ldots \cup h_n^{cn}$
  - The frontier of $C = \{e_i^{ci}, i = 1,2, \ldots n\}$

Consistent States

- A cut C is consistent if and only if
  - $\forall_{e \in C}$ (if $f \rightarrow e$ then $f \in C$)
- A global state S is consistent if and only if
  - it corresponds to a consistent cut



CSE 486/586 Administrivia

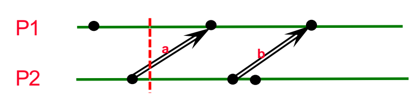
- PA2-A deadline: This Friday
- PA1: some hiccups, getting delayed
- Please come and ask questions during office hours.

The Snapshot Algorithm: Assumptions

- There is a communication channel between each pair of processes (@each process: N-1 in and N-1 out)
- Communication channels are unidirectional and FIFO-ordered (important point)
- No failure, all messages arrive intact, exactly once
- Any process may initiate the snapshot
- Snapshot does not interfere with normal execution
- Each process is able to record its state and the state of its incoming channels (no central collection)

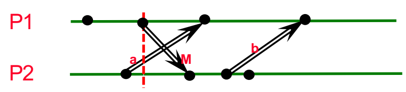
Reminder: Clock-Sync'd Snapshot

- Instantaneous snapshot
  - Process snapshots and network messages at time t
  - We can't quite do it due to imperfect clock sync.



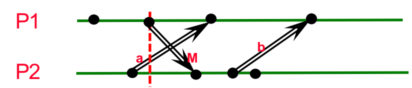
Chandy and Lamport's Snapshot: Basic Idea

- Goal: taking a consistent (not instantaneous) global snapshot
- Any process can initiate a snapshot-taking process by taking a local snapshot and sending a message called a marker.
- Upon receiving a marker, a process takes a local snapshot of its own.
- How do we capture network messages?
  - Insight: messages in flight will eventually arrive.



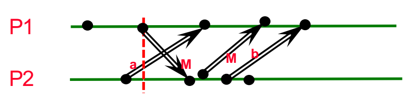
Chandy and Lamport's Snapshot: Basic Idea

- Each process that has taken a snapshot also starts recording incoming messages
  - Since those messages were in the network when the snapshot was being taken.
  - If every process does this, we will capture all messages in flight, recording messages destined to each process.
  - Note: every process needs to to this for every other process.
- Tricky part: the algorithm has a mechanism to stop recording incoming messages at some point.



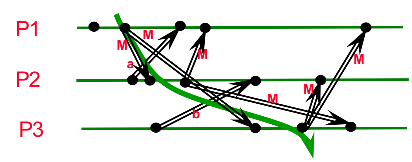
Chandy and Lamport's Snapshot: Basic Idea

- Reminder: which messages do we want to record?
  - Messages that were in the network at the time of taking a snapshot
- How do we record just those messages?
  - Insight: we can mark the end of relevant messages.
- After taking a local snapshot, each process sends a message saying that it's done sending all messages relevant to the snapshot.
  - In fact, we don't need a different message type, we use the same marker message.



Chandy and Lamport's Snapshot

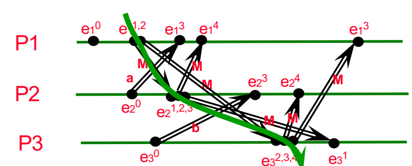
- Marker broadcast & recording
  - The initiator broadcasts a "marker" message to everyone else
  - If a process receives a marker for the first time, it takes a local snapshot, starts recording all incoming messages, and broadcasts a marker again to everyone else.
  - A process stops recording for each channel, when it receives a marker for that channel.



The Snapshot Algorithm

1. Marker sending rule for initiator process P_0
   - After P_0 has recorded its own state
     - for each outgoing channel C, send a marker message on C
2. Marker receiving rule for a process P_k
   on receipt of a marker over channel C
   - if P_k has not yet recorded its own state
     - record P_k's own state
     - record the state of C as "empty"
     - for each outgoing channel C, send a marker on C
     - turn on recording of messages over other incoming channels
   - else
     - record the state of C as all the messages received over C since P_k saved its own state; stop recording state of C

Exercise



1- P1 initiates snapshot: records its state (S1); sends Markers to P2 & P3; turns on recording for channels C21 and C31

2- P2 receives Marker over C12, records its state (S2), sets state(C12) = {} sends Marker to P1 & P3; turns on recording for channel C32

3- P1 receives Marker over C21, sets state(C21) = {a}

4- P3 receives Marker over C13, records its state (S3), sets state(C13) = {} sends Marker to P1 & P2; turns on recording for channel C23

5- P2 receives Marker over C32, sets state(C32) = {b}

6- P3 receives Marker over C23, sets state(C23) = {}

7- P1 receives Marker over C31, sets state(C31) = {}

One Provable Property

- The snapshot algorithm gives a consistent cut
- Meaning,
  - Suppose e_i is an event in P_i, and e_j is an event in P_j
  - If $e_i \rightarrow e_j$, and e_j is in the cut, then e_i is also in the cut.
- Proof sketch: proof by contradiction
  - Suppose e_j is in the cut, but e_i is not.
  - Since $e_i \rightarrow e_j$, there must be a sequence M of messages that leads to the relation.
  - Since e_i is not in the cut (our assumption), a marker should've been sent before e_i, and also before all of M.
  - Then P_j must've recorded a state before e_j, meaning, e_j is not in the cut. (Contradiction)

Summary

- Global states
  - A union of all process states
  - Consistent global state vs. inconsistent global state
- The "snapshot" algorithm
  - Take a snapshot of the local state
  - Broadcast a "marker" msg to tell other processes to record
  - Start recording all msgs coming in for each channel until receiving a "marker"
  - Outcome: a consistent global state

Acknowledgements

- These slides contain material developed and copyrighted by Indranil Gupta at UIUC.