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: An important algorithm related to the discussion of time

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

What is a Snapshot?
- Single process snapshot
  - Just a snapshot of the local state, e.g., memory dump, stack trace, etc.
- Multi-process snapshot
  - Snapshots of all process states
  - Network snapshot: All messages in the network

What Do We Want?
- Would you say this is a good snapshot?
  - “Good”: we can explain all the causality, including messages
  - No because e1 might have been caused by e2.
- Three things we want.
  - Per-process state
  - Messages that are causally related to each and every local snapshot and in flight
  - All events that happened before each event in the snapshot

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!
Obvious First Try

- Synchronize clocks of all processes
  - Ask all processes to record their states at known time \( t \)
- Problems?
  - Time synchronization possible only approximately
  - Another issue?
    - Does not record the state of messages in the channels
- Again: synchronization not required – causality is enough!
- What we need:
  - Logical global snapshot
    - The state of each process
    - Messages in transit in all communication channels

How to Do It? Definitions

- For a process \( P_i \), where events \( e_i^0, e_i^1, \ldots \) occur,
  - history(\( P_i \)) = \( h_i^0 = <e_i^0, e_i^1, \ldots> \)
  - prefix history(\( P_i \)) = \( h_i^k = e_i^0, e_i^1, \ldots, e_i^k > \)
- \( S_i^k \): \( P_i \)'s state immediately after \( k \)th event
- For a set of processes \( P_1, \ldots, P_i, \ldots \):
  - Global history: \( H = \bigcup_i h_i \)
  - Global state: \( S = \bigcup_i S_i \)
  - A cut \( C \subseteq H = h_1^{c_1} \cup h_2^{c_2} \cup \ldots \cup h_n^{c_n} \)
  - The frontier of \( C \) = \( \{ e_i^{c_i}, i = 1, 2, \ldots, n \} \)

Consistent States

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

Why Consistent States?

- #1: For each event, you can trace back the causality.
- #2: Back to the state machine (from the last lecture)
  - The execution of a distributed system as a series of transitions between global states: \( S_0 \rightarrow S_1 \rightarrow S_2 \rightarrow \ldots \)
  - \( \ldots \) where each transition happens with one single action from a process (i.e., local process event, send, and receive)
  - Each state (\( S_0, S_1, S_2, \ldots \)) is a consistent state.

CSE 486/586 Administrivia

- PA2-A deadline: This Friday
- 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)
Single Process vs. Multiple Processes

- Single process snapshot
  - Just a snapshot of the local state, e.g., memory dump, stack trace, etc.
- Multi-process snapshot
  - Snapshots of all process states
  - Network snapshot: All messages in the network

Two questions:
1. When to take a local snapshot at each process so that the collection of them can form a consistent global state? (Process snapshot)
2. How to capture messages in flight? (Network snapshot)

The Snapshot Algorithm

- Clock-synced snapshot (instantaneous snapshot)
  - Process snapshots and network messages at time \( t \)
  - Need to capture:
    - Local snapshots of \( P_1 \) & \( P_2 \)
    - Messages in the network (message \( a \), since message \( a \) is causally related to \( P_2 \)'s snapshot)
  - We can’t quite do it due to (i) imperfect clock sync and (ii) no help from the network.

- Logical snapshot (not instantaneous)
  - Goal: capturing causality (events and messages)
  - A process tells others to take a snapshot by sending a message (see the diagram).
  - But there’s a delay in doing so.
  - Need to capture all network messages during the delay (not at an instantaneous moment)
  - We need to capture:
    - Local snapshots of \( P_1 \) & \( P_2 \) (same as before but now at two different times).
    - Messages in flight that are causally related to each and every local snapshot, e.g., messages \( a \) and \( b \) for \( P_2 \)'s snapshot.
    - How?

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 \( C \)
   - 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.

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
   - Also
     - record the state of \( C \) as all the messages received over \( C \)
     - since \( P_k \) saved its own state; stop recording state of \( C \)
**Chandy and Lamport’s Snapshot**

Marker receiving rule for process $p_i$:
On $p_i$’s receipt of a marker message over channel $c$:
- If ($p_i$ has not yet recorded its state) it records its process state now;
  - turns on recording of messages arriving over other incoming channels;
  - $p_i$ records the state of $c$ as the empty set;
- Else
  - $p_i$ records the state of $c$ as the set of messages it has received over $c$ since it saved its state.

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 message over $c$).

**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 $e_i$ is in the cut, but $e_i$ is not.
  - Since $e_i \rightarrow e_j$, there must be a sequence $M$ of messages that lead to the relation.
  - Since $e_i$ is not in the cut (our assumption), a marker should have been sent before $e_i$, and also before all of $M$.
  - Then $P_i$ must have recorded a state before $e_i$, meaning, $e_i$ is not in the cut. (Contradiction)

**Exercise**

1. P1 initiates snapshot: records its state ($S_1$); sends markers to P2 & P3; turns on recording for channels C21 and C31
2. P2 receives marker over C12, records its state ($S_2$), sets state(C12) = {};
   sends marker to P1 & P3; turns on recording for channel C32
3. P1 receives marker over C21, sets state(C21) = {};
4. P3 receives marker over C13, records its state ($S_3$), sets state(C13) = {};
   sends marker to P1 & P2; turns on recording for channel C23
5. P2 receives marker over C32, sets state(C32) = {};
6. P3 receives marker over C23, sets state(C23) = {};
7. P1 receives marker over C31, sets state(C31) = {};

**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.