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 Topic
• Global snapshots
• For distributed programming, it’s important to be able to reason about your system’s behavior in the abstract.
• Today’s topic will further increase your understanding.

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
  – 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.
• Multi-process snapshot
  – Two things
  – Process snapshots: 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 e₁₂ might have been caused by e₁₁.

• 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

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 Pᵢ, where events eᵢ₀, eᵢ₁, … occur,
  – history(Pᵢ) = hᵢ = <eᵢ₀, eᵢ₁, …>
  – prefix history(Pᵢ) = Hᵢ = <eᵢ₀, eᵢ₁, …, eᵢₖ>
  – Sᵢₖ: Pᵢ’s state immediately after k⁰ event

• For a set of processes P₁, …, Pᵢ, …:
  – Global history: H = ∪ (hᵢ)
  – Global state: S = ∪ (Sᵢ)
  – A cut C ⊆ H = hᵢ₁, hᵢ₂, …, hᵢₙ
  – The frontier of C = {eᵢᵢ, i = 1, 2, …, n}

Consistent States

• A cut C is consistent if and only if
  ∀ e ∈ C (if f → e then f ∈ 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: The state machine view of a distributed system
  – The execution of a distributed system as a series of transitions between global states: S₀ → S₁ → S₂ → …
  – …where each transition happens with one single action from a process (i.e., local process event, send, and receive)
  – Each state (S₀, S₁, S₂, …) is a consistent state.

CSE 486/586 Administrivia

• PA2-A deadline: This Friday
• PA1: Will start grading from today
• 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.
  • But for the sake of this lecture, let’s say a log of all events
• 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)

Reminder: Clock-Sync’d Snapshot

• Instantaneous snapshot
  • Process snapshots and network messages at time t
  • We can’t quite do it due to (i) imperfect clock sync and (ii) no help from the network.

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. (The proc. snapshot part done)
  • Still need to take a network snapshot.
• How do we take a network snapshot?
  • 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.
• 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. \( P_1 \) initiates snapshot: records its state (\( S_1 \)); sends Markers to \( P_2 \) & \( P_3 \); turns on recording for channels \( C_{21} \) and \( C_{31} \)
2. \( P_2 \) receives Marker over \( C_{12} \), records its state (\( S_2 \)), sets state(\( C_{12} \)) = \{ \}
   sends Marker to \( P_1 \) & \( P_3 \); turns on recording for channel \( C_{32} \)
3. \( P_1 \) receives Marker over \( C_{21} \), sets state(\( C_{21} \)) = \{ a \}
4. \( P_3 \) receives Marker over \( C_{13} \), records its state (\( S_3 \)), sets state(\( C_{13} \)) = \{ \}
   sends Marker to \( P_1 \) & \( P_2 \); turns on recording for channel \( C_{23} \)
5. \( P_2 \) receives Marker over \( C_{32} \), sets state(\( C_{32} \)) = \{ b \}
6. \( P_3 \) receives Marker over \( C_{23} \), sets state(\( C_{23} \)) = \{ \}
7. \( P_1 \) receives Marker over \( C_{31} \), sets state(\( C_{31} \)) = \{ \}

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.