Logistics Related

• MP0 deadline extended by one day due to VM cluster service disruptions.
  • New due date: Friday, Feb 4th, 11:59pm.
Today’s agenda

• Global State

• Chapter 14.5

• Goal: reason about how to capture the state across all processes of a distributed system without requiring time synchronization.
Recap: How to capture global state?

• State of each process (and each channel) in the system at a given instant of time.
  • Difficult to capture -- requires precisely synchronized time.

• Relax the problem: find a consistent global state.
  • For a system with n processes \(<p_1, p_2, p_3, \ldots, p_n>\), capture the state of the system after the \(c_i^{th}\) event at process \(p_i\).
    • State corresponding to the cut defined by frontier events \(\{e_i^{c_i}, \text{for } i = 1, 2, \ldots n\}\).
  • We want the state to be consistent.
    • Must correspond to a consistent cut.
      • If an event \(e\) belongs to the cut, all events that “happened before” \(e\) must also belong to the cut.
Chandy-Lamport Algorithm Intuition

• First, initiator \( p_i \):
  • records its own state.
  • creates a special marker message.
  • sends the marker to all other process.
  • start recording messages received on other channels.
    • until a marker is received on a channel.

• When a process receives a marker.
  • If marker is received for the first time.
    • records its own state.
    • sends marker on all other channels.
    • start recording messages received on other channels.
      • until a marker is received on a channel.
Chandy-Lamport Algorithm: Properties

• Any run of the Chandy-Lamport Global Snapshot algorithm creates a consistent cut.
• Let $e_i$ and $e_j$ be events occurring at $p_i$ and $p_j$, respectively such that
  • $e_i \rightarrow e_j$ ($e_i$ happens before $e_j$)
• The snapshot algorithm ensures that
  • if $e_j$ is in the cut then $e_i$ is also in the cut.
• That is: if $e_j \rightarrow < p_j \text{ records its state}>$, then
  it must be true that $e_i \rightarrow < p_i \text{ records its state}>$. 
Chandy-Lamport Algorithm: Properties

- Given $e_i \rightarrow e_j$. If $e_j \rightarrow <p_j$ records its state>, then it must be true that $e_i \rightarrow <p_i$ records its state$>.

- By contradiction, suppose $e_j \rightarrow <p_j$ records its state>, and $<p_i$ records its state$> \rightarrow e_i$. 

![Diagram showing the relationships between processes and events](image-url)
Chandy-Lamport Algorithm: Properties

• Given $e_i \rightarrow e_j$. If $e_j \rightarrow <p_j$ records its state>, then it must be true that $e_i \rightarrow <p_i$ records its state$>.$

• By contradiction, suppose $e_j \rightarrow <p_j$ records its state$>$, and $<p_i$ records its state$> \rightarrow e_i.$
Chandy-Lamport Algorithm: Properties

- Given $e_i \rightarrow e_j$. If $e_j \rightarrow <p_j$ records its state>, then it must be true that $e_i \rightarrow <p_i$ records its state$>.

- By contradiction, suppose $e_j \rightarrow <p_j$ records its state$>$, and $<p_i$ records its state$> \rightarrow e_i$.

![Diagram showing time and events](image-url)
Chandy-Lamport Algorithm: Properties

• Given $e_i \rightarrow e_j$. If $e_j \rightarrow <p_j$ records its state>, then it must be true that $e_i \rightarrow <p_i$ records its state$>.

• By contradiction, suppose $e_j \rightarrow <p_j$ records its state$>$, and $<p_i$ records its state$> \rightarrow e_i$.

\begin{figure}[h]
\centering
\includegraphics[width=\textwidth]{chandy-lamport-diagram.png}
\caption{Diagram illustrating the Chandy-Lamport Algorithm properties.}
\end{figure}
Chandy-Lamport Algorithm: Properties

- Given $e_i \rightarrow e_j$. If $e_j \rightarrow <p_j \text{ records its state}>$, then it must be true that $e_i \rightarrow <p_i \text{ records its state}>$.
- By contradiction, suppose $e_j \rightarrow <p_j \text{ records its state}>$, and $<p_i \text{ records its state}> \rightarrow e_i$.
- Consider the path of app messages (through other processes) that go from $e_i$ to $e_j$.
- Due to FIFO ordering, markers on each link in above path will precede regular app messages.
- Thus, since $<p_i \text{ records its state}> \rightarrow e_i$, it must be true that $p_j$ received a marker before $e_j$.
- Thus $e_j$ is not in the cut => contradiction.
Summary

• The ability to calculate global snapshots in a distributed system is very important.
• But don’t want to interrupt running distributed application.
• Chandy-Lamport algorithm calculates global snapshot.
• Obey causality (creates a consistent cut).
• Can be used to detect global properties.
  • Safety vs. Liveness.
Chandy-Lamport Algorithm: Usefulness

• Consistent global snapshots are useful for detecting global system properties:
  • Safety
  • Liveness
More notations and definitions

- history(p_i) = h_i = \langle e_i^0, e_i^1, \ldots \rangle
- global history: H = \bigcup_i (h_i)

- A **run** is a total ordering of events in H that is consistent with each h_i’s ordering.
- A **linearization** is a run consistent with happens-before (\rightarrow) relation in H.
Example

Order at $p_1$: $< e_1^0, e_1^1, e_1^2, e_1^3 >$  
Order at $p_2$: $< e_2^0, e_2^1, e_2^2 >$

Causal order across $p_1$ and $p_2$: $< e_1^0, e_1^1, e_2^0, e_2^1, e_2^2, e_1^3 >$

Run: $< e_1^0, e_1^1, e_1^2, e_1^3, e_2^0, e_2^1, e_2^2 >$

Linearization: $< e_1^0, e_1^1, e_1^2, e_2^0, e_2^1, e_2^2, e_1^3 >$
Example

Order at $p_1$: $< e_1^0, e_1^1, e_1^2, e_1^3 >$  
Order at $p_2$: $< e_2^0, e_2^1, e_2^2 >$

Causal order across $p_1$ and $p_2$: $< e_1^0, e_1^1, e_2^0, e_2^1, e_2^2, e_1^3 >$

Run: $< e_1^0, e_1^1, e_1^2, e_1^3, e_2^0, e_2^1, e_2^2 >$
Linearization: $< e_1^0, e_1^1, e_1^2, e_2^0, e_2^1, e_2^2, e_1^3 >$
Example

Order at $p_1$: $<e_1^0, e_1^1, e_1^2, e_1^3>$  

Order at $p_2$: $<e_2^0, e_2^1, e_2^2>$

Causal order across $p_1$ and $p_2$: $<e_1^0, e_1^1, e_2^0, e_2^1, e_2^2, e_1^3>$

$<e_1^0, e_1^1, e_2^0, e_2^1, e_1^2, e_2^2, e_1^3>$
Order at $p_1$: $< e_1^0, e_1^1, e_1^2, e_1^3 >$  
Order at $p_2$: $< e_2^0, e_2^1, e_2^2 >$

Causal order across $p_1$ and $p_2$: $< e_1^0, e_1^1, e_2^0, e_2^1, e_2^2, e_1^3 >$

$< e_1^0, e_1^1, e_2^0, e_2^1, e_1^2, e_2^2, e_1^3 >$: Linearization

$< e_1^0, e_2^1, e_2^0, e_1^1, e_1^2, e_2^2, e_1^3 >$: Not even a run
More notations and definitions

• history(pᵢ) = hᵢ = ⟨eᵢ⁰, eᵢ¹, ... ⟩
• global history: H = ∪ᵢ (hᵢ)

• A run is a total ordering of events in H that is consistent with each hᵢ’s ordering.

• A linearization is a run consistent with happens-before (→) relation in H.

• Linearizations pass through consistent global states.
Example

Order at $p_1$: $<e_1^0, e_1^1, e_1^2, e_1^3>$  
Order at $p_2$: $<e_2^0, e_2^1, e_2^2>$

Causal order across $p_1$ and $p_2$: $<e_1^0, e_1^1, e_2^0, e_2^1, e_2^2, e_1^3>$

Linearization: $<e_1^0, e_1^1, e_1^2, e_2^0, e_2^1, e_2^2, e_1^3>$
Order at \( p_1 \): \(< e_1^0, e_1^1, e_1^2, e_1^3 >\)  
Order at \( p_2 \): \(< e_2^0, e_2^1, e_2^2 >\)  
Causal order across \( p_1 \) and \( p_2 \): \(< e_1^0, e_1^1, e_2^0, e_2^1, e_2^2, e_1^3 >\)  
Linearization: \(< e_1^0, e_1^1, e_1^2, e_2^0, e_2^1, e_2^2, e_1^3 >\)
Order at $p_1$: $\langle e_1^0, e_1^1, e_1^2, e_1^3 \rangle$

Order at $p_2$: $\langle e_2^0, e_2^1, e_2^2 \rangle$

Causal order across $p_1$ and $p_2$: $\langle e_1^0, e_1^1, e_2^0, e_2^1, e_2^2, e_1^3 \rangle$

Linearization: $\langle e_1^0, e_1^1, e_1^2, e_2^0 | e_2^1, e_2^2, e_1^3 \rangle$
Example

Order at $p_1$: $< e_1^0, e_1^1, e_1^2, e_1^3 >$    Order at $p_2$: $< e_2^0, e_2^1, e_2^2 >$

Causal order across $p_1$ and $p_2$: $< e_1^0, e_1^1, e_2^0, e_2^1, e_2^2, e_1^3 >$

Linearization: $< e_1^0, e_1^1, e_1^2, e_2^0, e_2^1, e_2^2, e_1^3 >$
Order at $p_1$: $<e_1^0, e_1^1, e_1^2, e_1^3>$  Order at $p_2$: $<e_2^0, e_2^1, e_2^2>$

Causal order across $p_1$ and $p_2$: $<e_1^0, e_1^1, e_2^0, e_2^1, e_2^2, e_1^3>$

Linearization: $<e_1^0, e_1^1, e_1^2, e_2^0, e_2^1, e_2^2, e_1^3>$
Example

Order at $p_1$: $< e_1^0, e_1^1, e_1^2, e_1^3 >$  
Order at $p_2$: $< e_2^0, e_2^1, e_2^2 >$  
Causal order across $p_1$ and $p_2$: $< e_1^0, e_1^1, e_2^0, e_2^1, e_2^2, e_1^3 >$

Linearization: $< e_1^0, e_1^1, e_1^2, e_2^0, e_2^1, e_2^2, e_1^3 >$
Example

Order at $p_1$: $< e_1^0, e_1^1, e_1^2, e_1^3 >$  
Order at $p_2$: $< e_2^0, e_2^1, e_2^2 >$

Causal order across $p_1$ and $p_2$: $< e_1^0, e_1^1, e_2^0, e_2^1, e_2^2, e_1^3 >$

Linearization: $< e_1^0, e_1^1, e_1^2, e_2^0, e_2^1, e_2^2, e_1^3 >$

Linearization $< e_1^0, e_1^1, e_2^0, e_2^1, e_1^2, e_2^2, e_1^3 >$
More notations and definitions

• Linearizations pass through consistent global states.

• A global state $S_k$ is reachable from global state $S_i$, if there is a linearization that passes through $S_i$ and then through $S_k$.

• The distributed system evolves as a series of transitions between global states $S_0, S_1, ...$. 
Many linearizations:

- $< p_0, p_1, p_2, q_0, q_1, q_2 >$
- $< p_0, q_0, p_1, q_1, p_2, q_2 >$
- $< q_0, p_0, p_1, q_1, p_2, q_2 >$
- $< q_0, p_0, p_1, p_2, q_1, q_2 >$
- $\ldots$
State Transitions: Example

**Execution Lattice.** Each path represents a linearization.
State Transitions: Example

Execution Lattice. Each path represents a linearization.
State Transitions: Example

**Execution Lattice.** Each path represents a linearization.
State Transitions: Example

Execution Lattice. Each path represents a linearization.
State Transitions: Example
State Transitions: Example
State Transitions: Example

[start] → $s_{0,0}$ → $s_{1,0}$ → $s_{2,0}$ → $s_{3,0}$ → $s_{2,1}$ → $s_{3,1}$ → $s_{3,2}$ → $s_{3,3}$

$p_0$ $p_1$ $p_2$ $q_0$ $q_1$ $q_2$
State Transitions: Example
More notations and definitions

• A **run** is a total ordering of events in H that is consistent with each \( h_i \)'s ordering.

• A **linearization** is a run consistent with happens-before (\( \rightarrow \)) relation in H.

• Linearizations pass through consistent global states.

• A global state \( S_k \) is reachable from global state \( S_i \), if there is a linearization that passes through \( S_i \) and then through \( S_k \).

• The distributed system evolves as a series of transitions between global states \( S_0, S_1, \ldots \).
Global State Predicates

• A global-state-predicate is a property that is *true* or *false* for a global state.
  • Is there a deadlock?
  • Has the distributed algorithm terminated?

• Two ways of reasoning about predicates (or system properties) as global state gets transformed by events.
  • Liveness
  • Safety
Liveness

- **Liveness** = guarantee that something **good** will happen, **eventually**

**Examples:**
- A distributed computation will terminate.
- “Completeness” in failure detectors: the failure will be detected.
- All processes will eventually decide on a value.

A global state $S_0$ satisfies a **liveness** property $P$ iff:
- $\text{liveness}(P(S_0)) \equiv \forall L \in \text{linearizations from } S_0, \ L \text{ passes through a } S_L \text{ & } P(S_L) = \text{true}$
- For all linearizations starting from $S_0$, $P$ is true for some state $S_L$ reachable from $S_0$. 
Liveness Example

If predicate is true only in the marked states, does it satisfy liveness? Yes
Liveness Example

If predicate is true only in the marked states, does it satisfy liveness? **No**
Liveness Example

If predicate is true only in the marked states, does it satisfy liveness? Yes
Liveness

- **Liveness** = guarantee that something **good** will happen, eventually

- **Examples:**
  - A distributed computation will terminate.
  - “Completeness” in failure detectors: the failure will be detected.
  - All processes will eventually decide on a value.

- A global state $S_0$ satisfies a **liveness** property $P$ iff:
  - $\text{liveness}(P(S_0)) \equiv \forall L \in \text{linearizations from } S_0, \ L \text{ passes through a } S_L \land P(S_L) = \text{true}$
  - For any linearization starting from $S_0$, $P$ is true for **some** state $S_L$ reachable from $S_0$. 
Safety

- **Safety** = guarantee that something **bad** will **never** happen.

- **Examples:**
  - There is no deadlock in a distributed transaction system.
  - “Accuracy’’ in failure detectors: an alive process is not detected as failed.
  - No two processes decide on different values.

- A global state $S_0$ satisfies a safety property $P$ iff:
  - $\text{safety}(P(S_0)) \equiv \forall S \text{ reachable from } S_0, P(S) = \text{true}$.
  - For all states $S$ reachable from $S_0$, $P(S)$ is true.
Safety Example

If predicate is true only in the marked states, does it satisfy safety? \(\textbf{No}\)
Safety Example

If predicate is true only in the **unmarked** states, does it satisfy safety?

**Yes**
Safety

• Safety = guarantee that something bad will never happen.

• Examples:
  • There is no deadlock in a distributed transaction system.
  • “Accuracy” in failure detectors: an alive process is not detected as failed.
  • No two processes decide on different values.

• A global state $S_0$ satisfies a safety property $P$ iff:
  • $\text{safety}(P(S_0)) \equiv \forall S \text{ reachable from } S_0, P(S) = \text{true}$.
  • For all states $S$ reachable from $S_0$, $P(S)$ is true.
Liveness Example

Technically satisfies liveness, but difficult to capture or reason about.
Stable Global Predicates

• once true, stays true forever afterwards (for stable liveness)
Stable Global Predicates

If predicate is true only in the marked states, is it stable?

No
Stable Global Predicates

If predicate is true only in the marked states, is it stable?

No
Stable Global Predicates

If predicate is true only in the marked states, is it stable? Yes
Stable Global Predicates

- once true, stays true forever afterwards (for stable liveness)
- once false, stays false forever afterwards (for stable non-safety)
- Stable liveness examples (once true, always true)
  - Computation has terminated.
- Stable non-safety examples (once false, always false)
  - There is no deadlock.
  - An object is not orphaned.

- All stable global properties can be detected using the Chandy-Lamport algorithm.
Global Snapshot Summary

• The ability to calculate global snapshots in a distributed system is very important.
• But don’t want to interrupt running distributed application.
• Chandy-Lamport algorithm calculates global snapshot.
• Obeys causality (creates a consistent cut).
• Can be used to detect global properties.
• Safety vs. Liveness.