Global States and Snapshots
Key Properties

- Multiple computers
  - Concurrent execution
  - Independent failures
  - Autonomous administrators
  - Heterogeneous capacities, properties
  - Large numbers (scalability)
- Networked communication
  - Asynchronous execution
  - Unreliable delivery
  - Insecure medium
- Common goal
  - Consistency – can discuss whole-system properties
  - Transparency – can use the system without knowing details
Outline

- Global States
  - Motivation
  - Definition
- Snapshots
  - Chandy-Lamport algorithm
Global State

- Want to find out a *global* property of the system
  - but can only make local observations at any time
Global State: Count

- Count the number of people in all EWS labs
- Option 1:
  - Go to each lab and count up people
  - Problems?
- Option 2:
  - Send one person to each lab to count
  - Problems?

\[ \text{Lab 1} @ 12:00 \]
\[ \text{Lab 2} @ 12:05 \]
\[ \text{Synchonize} \]
Distributed Garbage Collection

- Newspaper on kitchen counter
  - Should you recycle it?
- Go to each roommate’s room, ask if roommate done
  - When can you throw out the paper?
- Stable property
  - If someone says “not done”, may not be safe to recycle
  - If everyone says “done”, must be safe to recycle
Other Applications

- Detect other stable properties
  - Termination
  - Deadlock
- Save a checkpoint
  - Recover to a “known good” state
Histories

\[ p_1 \rightarrow e^0_p \rightarrow S_1 \rightarrow e^1_1 \rightarrow S_1 \rightarrow \]

\[ p_2 \rightarrow e^0_2 \rightarrow S_2 \rightarrow e^1_2 \rightarrow S_2 \rightarrow \]

\[ p_3 \rightarrow \]
Process Histories and States

- For a process $P_i$, where events $e_i^0, e_i^1, \ldots$ occur:
  
  $\text{history}(P_i) = h_i = \langle e_i^0, e_i^1, \ldots \rangle$

  $\text{prefix history}(P_i^k) = h_i^k = \langle e_i^0, e_i^1, \ldots, e_i^k \rangle$

  $S_i^k : P_i$’s state immediately after $k$th event

- For a set of processes $P_1, \ldots, P_i, \ldots$:

  $\text{global history}: H = \bigcup_i (h_i)$

  $\text{global state}: S = \bigcup_i (S_i^k)$

  a cut $C \subseteq H = h_1^{c_1} \cup h_2^{c_2} \cup \ldots \cup h_n^{c_n}$

  the frontier of $C = \{e_i^{c_i}, i = 1, 2, \ldots, n\}$
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

A consistent cut == a global snapshot
Chandy-Lamport Algorithm

- **Goal:** Record global snapshot
  - Process state
  - Channel state
  - Consistent cut
- **System model**
  - FIFO, reliable channels between any two processes
  - Each process records own state (no central collection)
  - Any process may initiate algorithm
Contagion

- Two colors for processes / events / channels
  - Blue: before snapshot
  - Red: after snapshot
- To initiate:
  - Color self red
  - Send red marker to each other process
- When red marker received in blue state
  - Color self red
  - Send red marker to each other process
Example

P1: a \rightarrow b \rightarrow c \rightarrow d

P2: e \rightarrow f \rightarrow g \rightarrow h

P3: i \rightarrow j

initiate snapshot

Nikita Borisov — U. Illinois
Example
Example

initiate snapshot

P1
a b c d
P2
e f g
P3
h i

2019-01-24
Nikita Borisov — U. Illinois
Example
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;
records the state of $c$ as the empty set;
turns on recording of messages arriving over other incoming channels;

Else

$p_i$ records the state of $c$ as the set of messages it has received over $c$ since it saved its state.

End if

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 $c$).

Note: initiating the snapshot is like receiving a marker from yourself
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) = {”}
Habitrail
Pod 1 has 3 hamsters

Pod 1

Pod 2

Pod 3

Nikita Borisov — U. Illinois
Channel State

Pod 1

Channel 2 -> 1 has 1 hamster

Pod 2

Channel 3 -> 1 is empty

Pod 3

Nikita Borisov — U. Illinois
Events

Pod 1

Hamster dies in Pod 1

Hamster born in Pod 2

Pod 2

Hamster leaves on channel 1->3

Pod 3

Hamster arrives on channel 3->2

Hamster arrives on channel 3

Hamster leaves on channel 1

Nikita Borisov — U. Illinois
Pod 1
- 3 hamsters

Channel 2->1
- empty

Pod 2
- 1 hamster

Channel 3->2
- 1 hamster

Pod 3
- 1 hamster

Channel 1->3
- 1 hamster

Channel 1->2
- empty

Channel 3->1
- empty

Pod 3 state:
- 1 hamster

Pod 2 state:
- 3 hamsters
Let $e_i$ and $e_j$ be events occurring at $p_i$ and $p_j$, respectively such that $e_i \rightarrow e_j$.

The snapshot algorithm ensures that if $e_j$ is in the cut then $e_i$ is also in the cut.

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 $<p_i \text{ records its state}> \rightarrow e_i$
- Consider the path of app messages (through other processes) that go from $e_i \rightarrow e_j$
- Due to FIFO ordering, markers on above path 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
Global States useful for detecting Global Predicates

- A cut is consistent if and only if it does not violate causality.
- 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, L, that passes through $S_i$ and then through $S_k$.
- The distributed system evolves as a series of transitions between global states $S_0, S_1, \ldots$. 

$e_0^0, e_1^0, e_2^0, e_7^0, e_1^1, e_3^0$
Global State Predicates

- A global-state-predicate is a function from the set of global states to \{\text{true, false}\}, e.g., deadlock, termination

- A global state \(S_0\) satisfies a liveness property \(P\) iff:
  \[
  \text{liveness}(P(S_0)) \equiv \forall L \in \text{linearizations from } S_0 \quad L \text{ passes through a } S_L \land P(S_L) = \text{true}
  \]

- Ex: \(P(S) = \text{the computation will terminate from } S\)

- A global state \(S_0\) satisfies a safety property \(P\) if:
  \[
  \text{safety}(P(S_0)) \equiv \forall S \text{ reachable from } S_0, \quad P(S) = \text{true}
  \]

- Ex: \(P(S) = S \text{ has no deadlock}\)

- Global states useful for detecting stable global-state-predicate: it is one that once it becomes true, it remains true in subsequent global states, e.g., an object \(O\) is orphaned, or deadlock

  - A stable predicate may be a safety or liveness predicate
Quick Note – Liveness versus Safety

- Can be confusing, but terms are very important:
  - Liveness = guarantee that something good will happen, eventually
    - “Guarantee of termination” is a liveness property
    - Guarantee that “at least one of the athletes in the 100m final will win gold” is liveness
    - A criminal will eventually be jailed
    - Completeness in failure detectors
  - Safety = guarantee that something bad will never happen
    - Deadlock avoidance algorithms provide safety
    - A peace treaty between two nations provides safety
    - An innocent person will never be jailed
    - Accuracy in failure detectors