Global States

CSE 486: Distributed Systems

Ethan Blanton

Department of Computer Science and Engineering

University at Buffalo

Recording State

Recording the state of a system has many uses:

- Laptop hibernation
- Process core dumps
- Filesystem snapshots
- Database checkpoints
- Debugging

In each of these cases, the state must be logically instantaneous.

For a single system, typically this can be simulated.

Distributed State

In an asynchronous distributed system, instantaneous is hard.

A perfect global physical clock would help:

- Every process can save its state simultaneously.
- What about messages?

Remember that messages can be arbitrarily delayed!

Logical Clocks

Causality can help.

What if we:

- don’t worry about a perfect simultaneous snapshot, but
- record a state that could have happened?

That is, happens before is perfectly preserved.

A Consistent Global State

We want to record the state of all processes such that:

- The internal state of all processes is preserved
- The messages “in flight” are preserved
- The recorded state captures a possible global state

Note that the consistent state may never have actually occurred.

Given a deterministic algorithm, restarting the system from this state should reach the same result as the actual system that was recorded.

Process Model

A process P is a series of events $p_0, \ldots, p_n$.

The preserved process state is all events $p_0, \ldots, p_i$ for some $0 \leq i \leq n$.

Sending a message m from P to Q is an event $s = p_i$.

Receiving the message m is an event $r = q_j$.

A message is an event such that $s \rightarrow m \rightarrow r$.

Processes send messages on channels.

Channel Model

A channel is a unidirectional, in-order communication mechanism.

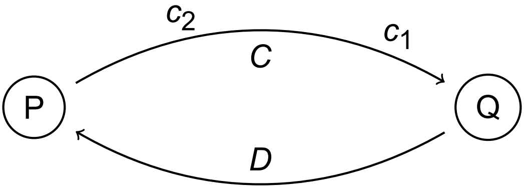
Assume that a channel C carries messages from P to Q.

C carries a series of messages $c_0, \ldots, c_n$.

The preserved state of C is a (possibly empty) set of messages $c_i, \ldots, c_j$, such that:

- The last state preserved by Q is q.
- c_i is the first message received by Q on C after q.
- The last state preserved by P is p.
- c_j is the last message sent by P on C before p.

A Snapshot



Global State:

P state $p_0, \ldots, p_i$, Q state $q_0, \ldots, q_j$

C state $\{c_1, c_2\}$, D state $\{\}$

Happens Before

This state preserves the happens before relationship.

Given a global state $S = \langle \{P\}, \{C\} \rangle$ where:

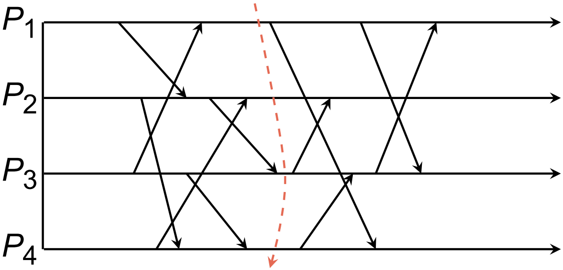
- $\{P\}$ is a set of processes states $P_0, \ldots, P_n$
- $\{C\}$ is a set of channel states $C_0, \ldots, C_n$

Let E be the set of all captured events in $\{P\}$ and $\{C\}$.

For each $e \in E$, for every $e' \to e$, $e' \in E$.

Consistent Cut

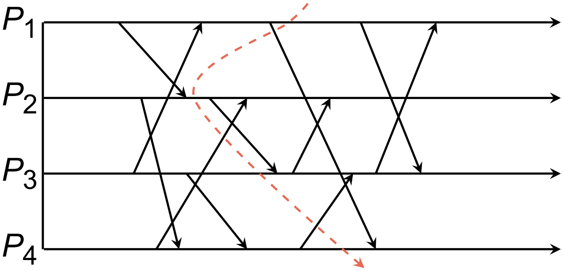
A consistent cut is a cut of events that preserves happens-before.



Sometimes trivial …

Consistent Cut

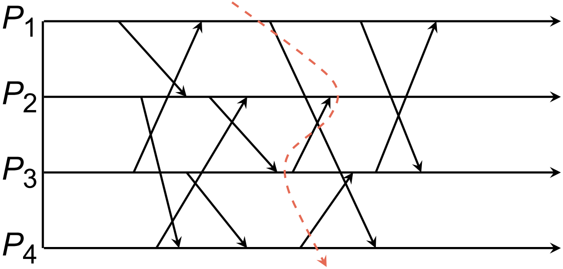
A consistent cut is a cut of events that preserves happens-before.



Sometimes less so!

Inconsistent Cut

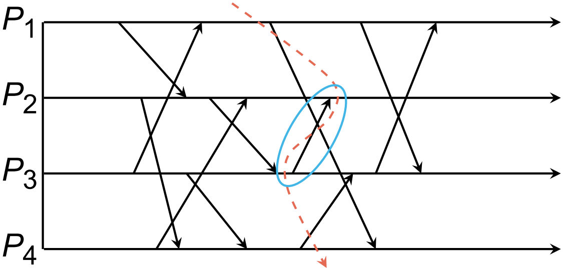
An inconsistent cut violates happens-before.



This cut is inconsistent — why?

Inconsistent Cut

An inconsistent cut violates happens-before.



This cut is inconsistent — why?

Chandy-Lamport

The Chandy-Lamport algorithm [1] records global states.

It operates by sending extra messages to initiate a snapshot.

It does not handle collecting the data from each process.

Any process may start a snapshot at any time.
(Even simultaneously!)

Assumptions

The Chandy-Lamport algorithm assumes:

- No process fails during the snapshot.
- Every process participates in finite time.
- No messages are lost.
- Every message is delivered in finite time.
- Communication channels are process-pairwise and unidirectional.
- Messages on a communication channel are delivered in-order.

Messages need not be globally in-order.

Markers

The extra messages sent are called markers.

Markers are separate from the processes' normal communication.

Markers are not recorded in the snapshot.

A marker's place in a channel bounds the snapshot.

Markers both:

- Trigger a process to take a snapshot itself
- Serve as notification that another process has taken a snapshot

The Algorithm

Marker-Sending Rule for a Process *p* [1]:

1. *p* records its state.
2. For each channel *c* outgoing from *p*:
   p sends one marker on *c* before sending any other messages on *c*

Marker-Receiving Rule for a Process *q*:
Upon receiving a marker on a channel *c*:

1. If *q* has not recorded its state, *q* executes the Marker-Sending Rule.
2. If *q* has recorded its state:
   q records every message received on *c* since it recorded its state.

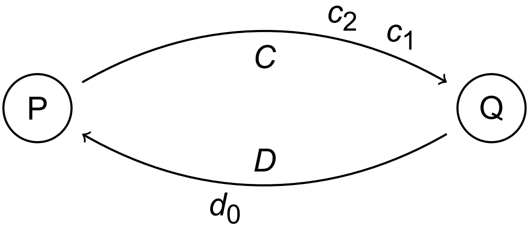
Operation

That’s it. That’s the whole thing.

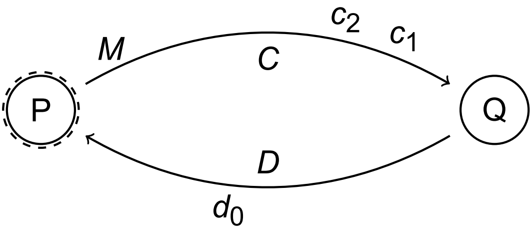
Once any process executes the Marker-Sending Rule, it starts.

Once every process has received a marker on every channel, it’s done.

How it Works

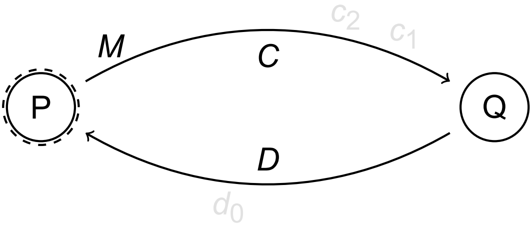


How it Works



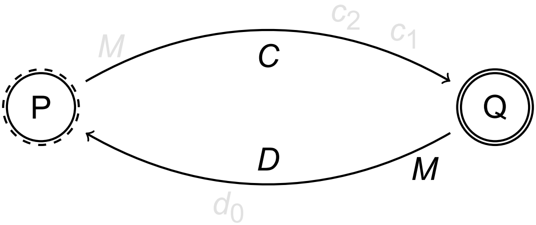
- *P* records *P*’s state and sends *M* on *C*

How it Works



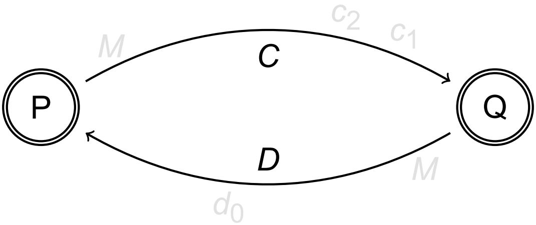
- P records P's state and sends M on C
- Q processes c_1 and c_2, P records d_0 on D

How it Works



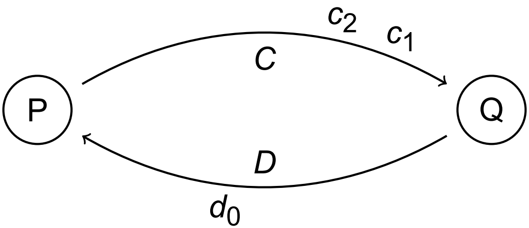
- P records P's state and sends M on C
- Q processes c_1 and c_2, P records d_0 on D
- Q receives M on C, records its state, sends M, and finishes

How it Works



- P records P's state and sends M on C
- Q processes c_1 and c_2, P records d_0 on D
- Q receives M on C, records its state, sends M, and finishes
- P receives M on D and finishes

How it Works



- State of P before d_0
- State of Q including c_1, c_2
- Q stores $C = \{\}$
- P stores $D = \{d_0\}$

Intuition

Think of the messages like light expanding outward.

Processes and messages in the light have been captured.

Processes and messages in the dark have not yet.

Because channels are FIFO, this ensures a consistent cut.

This is like special relativity!

Summary

- Global states are useful for many purposes
- A consistent global state could have happened
- Consistency is ensured by preserving happens before
- Chandy-Lamport snapshots capture global state
  - More work is needed without reliable, ordered messages

References I

Required Readings

[2] Ajay D. Kshemkalyani and Mukesh Singhal. *Distributed Computing: Principles, Algorithms, and Systems*. Chapter 4: Intro, 4.1–4.3. Cambridge University Press, 2008. ISBN: 978-0-521-18984-2.

Optional Readings

[1] K. Mani Chandy and Leslie Lamport. “Distributed Snapshots: Determining Global States of Distributed Systems”. In: *ACM Transactions on Computing Systems* 3.1 (Feb. 1985), pp. 63–75. URL: `https://lamport.azurewebsites.net/pubs/chandy.pdf`.

References II

[3] Friedemann Mattern. “Virtual Time and Global States of Distributed Systems”. In: *Proceedings of the Workshop on Parallel and Distributed Algorithms*. Elsevier Science Publishers B.V., Oct. 1988, pp. 215–226. URL: `http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.1068.1331`.

Copyright 2021–2024 Ethan Blanton, All Rights Reserved.

Reproduction of this material without written consent of the author is prohibited.

To retrieve a copy of this material, or related materials, see `https://www.cse.buffalo.edu/~eblanton/`.