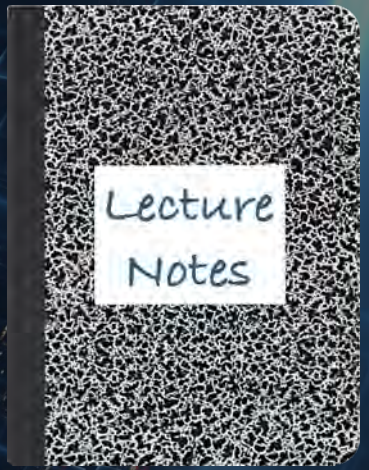
CS 417 – DISTRIBUTED SYSTEMS

Week 3: Hybrid Logical Clocks Discussion



Paul Krzyzanowski

Hybrid Logical Clocks

Goal of Hybrid Logical Clocks:

Combine the advantages of
physical clocks (tracking real time)
with **logical clocks** (track causality)

Weaknesses of Physical clocks

- Time may drift or jump backward
- Lack of perfect synchronization can hide the order of events
- Clock resolution can hide the order of events

Weakness of Logical clocks

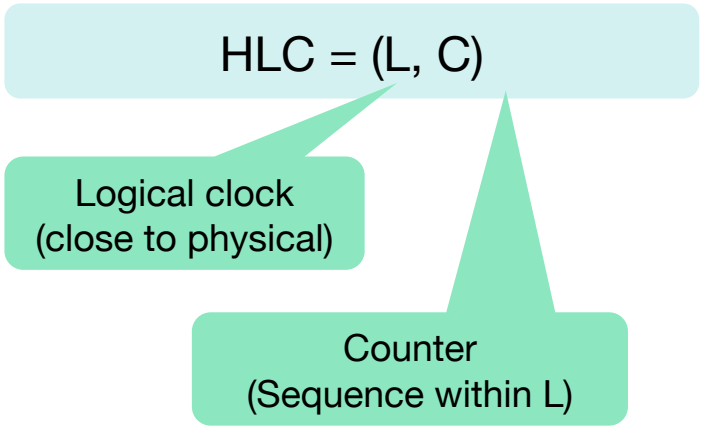
- Lamport clocks track causality (A happened before B), but have no relation to real-world time

Benefits of Hybrid Logical Clocks

- **Capture causal relationships**
  If $a \rightarrow b$ (event *a* caused event *b),* then the timestamp of event *a* is less than *b*
- **Keep proximity to physical time**
  Keep the logical clock value close to physical time (NTP), allowing it to be used for time-based queries in databases
- **Improve data consistency at scale**
  HLCs provide total ordering of events, which is crucial for distributed databases to ensure data consistency without relying on perfectly synchronized clocks
- **Resilient to jumps and drift**
  The logical clock handles ordering even when physical clocks are inaccurate or jump backwards
- **Low overhead solution**
  Unlike vector clocks, the size is a timestamp + a small counter

Hybrid Logical Clocks (HLC) Structure

- Assumptions
  - Every system has a **physical clock** , P, that is coarsely synchronized to the real time (via NTP, for example)
  - Every process will maintain an HLC = (L, C)
    - L (logical clock): a timestamp that will be close to the physical clock
    - C: a small counter
  - Events are timestamped with the HLC
  - Every message contains the HLC
- Comparing timestamps: $HLC_1 < HLC_2$ if
  - $HLC_1.L < HLC_2.L$
  - or $HLC_1.L == HLC_2.L$ and $HLC_1.C < HLC_2.C$



Example: Local Events

HLC = (L, C)
L will never be smaller than the physical timestamp

1. If the physical time advances, set L to the new time
2. If it doesn't, increment the counter to keep event sequences consistent

- Suppose L=1000, C=5
- Steps:
  1. Read physical time P
  2. if (P > L)
     L = P
     C = 0

P = 1001, HLC = (1000, 0)

New event timestamp:
Is (P > L)? Yes.
- Set L = 1001
- Set C = 0

New timestamp: HLC = (1001, 0)

Example: Receive message with larger timestamp

We need to preserve the relationship *a→b* where *a=msg sent, b=msg received*
L should never be smaller than the received timestamp

1. If the physical timestamp is greater than any L, new time = (P, 0)
2. If the L in the received timestamp is greater:
   a. Set L = received L
   b. Set C = received C + 1
3. If L in the received message is the same, C should reflect the order
   – Set C = max(C, received C) + 1

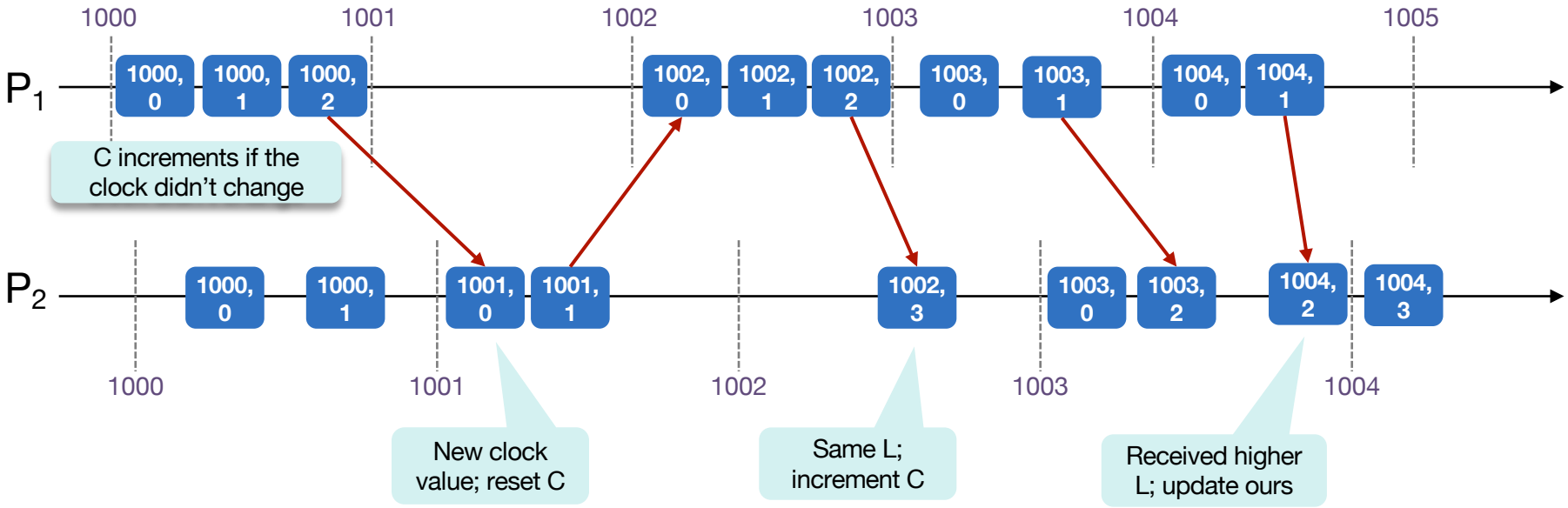
P = 1001, HLC = (1000, 0)
Receive (1005, 4)

New event timestamp:
Is ($P > L$ and $P > L_{msg}$)? No.
Is ($L_{msg} > L$)? Yes.
– Set L = 1005
– Set C = 4+1
New timestamp: HLC = (1005, 5)

Example

Here's an example of how HLCs can look for a set of events.

- We have two processes, P_1 and P_2, that generate events and exchange messages.
- Each event will be shown with its (L, C) timestamp.
- The vertical bars represent the ticking (and resolution) of the physical clock on each system.
- Physical clocks are closely synchronized but not perfectly, since they drift.



Who uses HLCs?

- HLCs are useful for applications that need wall time (real-world time)

  … *AND* a Lamport-style causality guarantee

  … without the space cost of adding vector clocks or using specialized hardware to ensure ultra-synchronized, high-resolution clocks

- Main applications are:
  - Database versioning, time-to-live values, time-based queries
- Some places where it's used:
  - CockroachDB: transaction timestamps, object versions, ordering
  - YugabyteDB: transaction timestamps, coordination
  - MongoDB: consistency, queries

When to use which clocks?

- **Cristian/NTP:**
  Keep system clocks close enough to UTC for timeouts, logs, leases, and expirations.
- **Lamport**
  reserve causal ordering cheaply, but cannot detect concurrency.
- **Vector clocks**
  Detect concurrency, good for conflict detection, but can get large.
- **HLC**
  Practical middle ground: causal ordering plus near-physical timestamps. Preserves causality but does NOT prove concurrency.

The End