HYBRID CLOCKS FOR HIGH AUDITABILITY

Murat Demirbas
University at Buffalo, SUNY

Auditability

- For distributed systems, naive logging is insufficient. Dapper, Zipkin, X-Trace perform sparse logging & cannot give a global consistent snapshot for any query time.

- Theory of distributed systems shunned the notion of time & used logical clocks (LC) to capture event ordering: E hb F ⇒ LC.E < LC.F   Using LC, it is not possible to query events in relation to real time.

- Practical distributed systems employed NTP synchronized clocks to capture time but in ad hoc undisciplined ways. Due to sync errors, we may have pt.E > pt.F when E hb F.

Our proposal

- By leveraging hybrid logical clocks (HLC) & hybrid vector clocks (HVC) as building blocks, we aim to bridge time + causality and design highly auditable distributed systems.

- HLC and HVC provide consistent-state snapshots efficiently without waiting-out time sync uncertainty & requiring prior coordination.

- HLC & HVC are always nonblocking & correct (albeit with reduced efficiency) even when time sync has degraded.

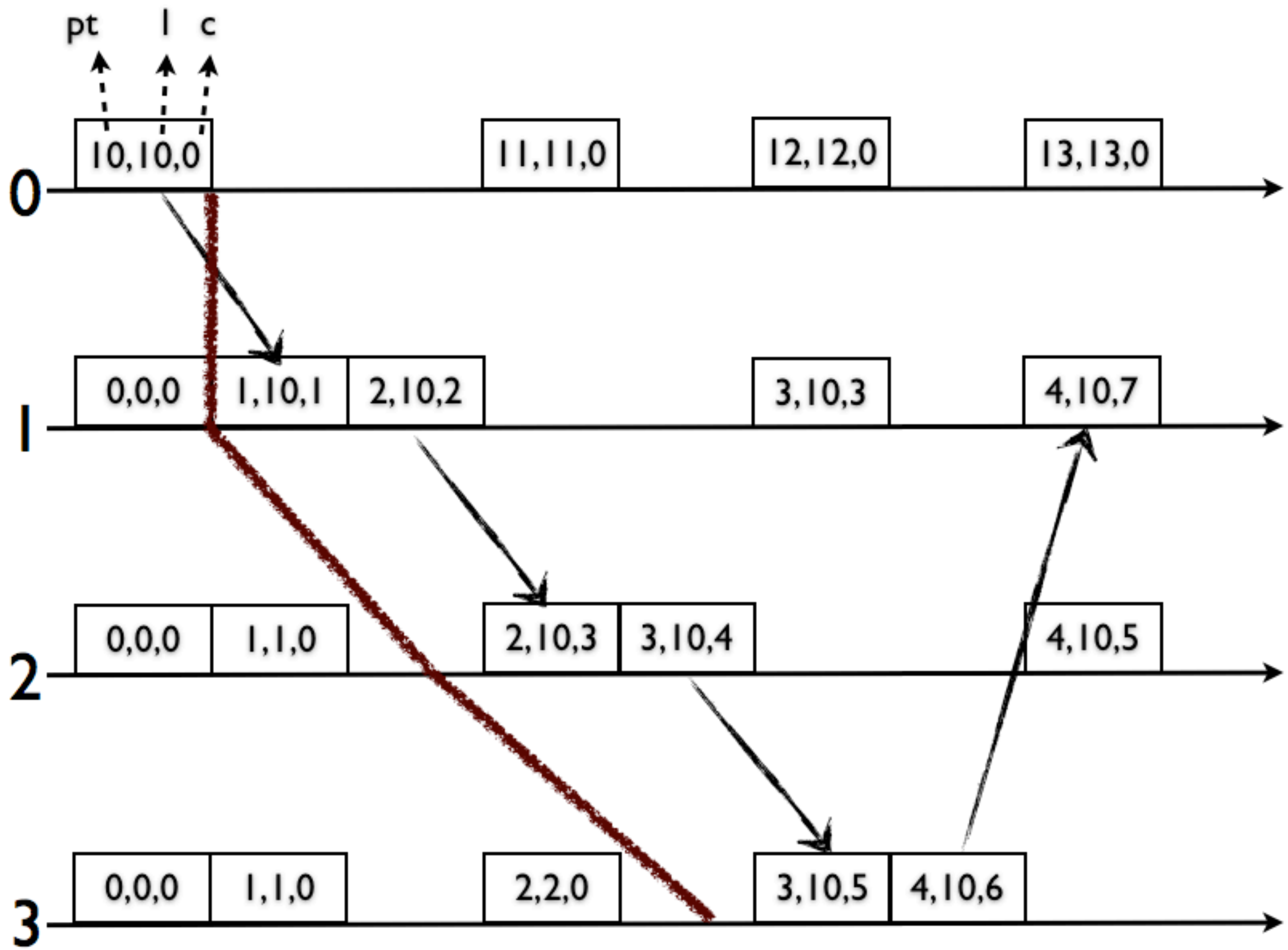
HLC implementation

- Each node j maintains timestamp of the form ⟨l.j,c.j⟩
  - pt.j corresponds to the physical clock of process j
  - l.j denotes the maximum pt.k that j is aware of
  - c.j captures the length of the causal chain

- Given a maximum clock drift of ε, HLC guarantees that l.j is in the range [pt.j, pt.j + ε]

- c.j is bounded, because c.j is reset to 0 when l.j increases (which inevitably happens in the worst case when pt.j exceeds l.j). Theoretically, the bound on c.j is proportional to the number of processes and ε. Practically, c.j is always less than 10 even under stress testing over AWS.

HLC makes consistent snapshots easy



A consistent snapshot for pt=10 using HLC. Here ε = 10.

HLC vs TrueTime (TT) in Google Spanner

- Since TT requires waiting-out uncertainty windows for the transaction commit, ε throttles the throughput of read-write transactions on a tablet level.

- The HLC-based implementation avoids waiting-out ε, instead records causality within this uncertainty window. HLC avoids the need for dedicated GPS/atomic clock & can work with NTP with ε of several tens of milliseconds.

- Our HLC clocks are adopted by CockroachDB, an opensource clone of Google Spanner.

HVC implementation

- Worst case: HVC at node j (i.e. hvc.j) is a vector containing one entry for each node: hvc.j[j] is the wallclock at j & hvc.j[k] is the knowledge of j about the wallclock of node k.
- Using loosely synced (ε) clocks, HVC becomes efficient. If j does not hear (directly or transitively) from k within ε then hvc.j[k] need not be explicitly maintained. We still infer that hvc.j[k] equals hvc.j[j]−ε thanks to clock sync.
- The size of hvc.j depends only on the number of nodes that communicated with j within the last ε and provided a fresh timestamp that is higher than hvc.j[j] − ε.

HVC bounds

- ε: uncertainty window, δ: minimum message delay, α: message rate, and n: number of nodes.
- HVC size is a sigmoid function wrt increasing ε; it has a slow start but it grows exponentially after a critical phase. We derive this threshold as $(1/\alpha + \delta)(\ln((2-\sqrt{3})(n-1)))$.
  - Even using aggressive α & δ, transition occurs at 2 sec >> ε
- For all practical applications & environments, the size of HVC remains only as a couple entries. Yet, when it is needed HVC expands on-demand to allow more entries to capture causality both ways in the ε uncertainty slices.

HVC for debugging

- HVC satisfy the vector clock comparison condition, and can serve in applications that HLC become inadequate.
- In contrast to HLC that provide a single consistent snapshot for a given time, HVC provide all possible/ potential consistent snapshots for that time.
- HVC find applications in debugging for concurrency race conditions of safety critical distributed systems and in causal delivery of messages to distributed system nodes.