Lamport Clocks, Time, and Ordering Events

15-712 #4
Fall 2007

Announcements

- Waitlist processed. If you're attending class today, you're probably in the course. :)
- New project Wiki page set up. See web page for details. Use for coordinating, finding partners, discussing ideas, etc.

Today's Star

- Time, Clocks, and the Ordering of Events in a Distributed System
  - Leslie Lamport
- PODC Influential Paper, 2000

Why's it cool?

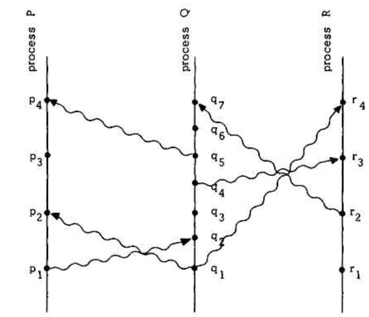
- Time & ordering are core to distributed systems logic
- Getting it wrong is a common and classic source of errors
  - Really nasty errors
    - Heisenbugs, Performance bugs, Porting bugs
- Formalizes a way to correctly implement a distributed state machine
  - In other words, *just about anything*

Causal Ordering

- Events may not be ordered
- "Before" and "After" abstractions usually wrong
- Need a causal link for "happened before"
  - The ordering of events is really a partial ordering
- True for multithreading, multi-programming
  - Even a single node has simultaneity problems

Looking at an ordering

- Simultaneous: No causal path up space/time diagram
- The set of "happens before" arcs for a specific run is unique
- Permit out of order message arrival
  - q1 -> r4
  - q4 -> r3

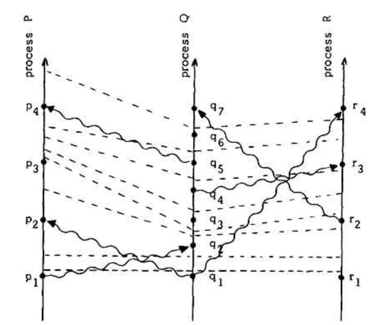


Logical Clocks

- Assign #s to events
  - If there is a causal path from A to B
    - C(A) < C(B) for all events A, B
  - Note: Says nothing about order of other events
    - Can implement arbitrary tie-breakers
    - (Which may affect important properties like fairness)

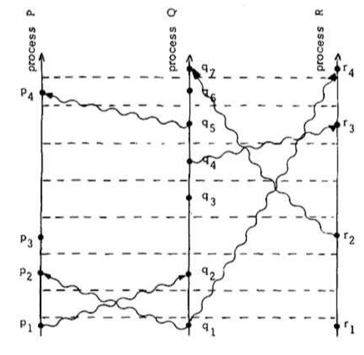
Looking at Logical Clock

- Add ticks btwn events in one 'process' (thread)
- Ticks crossing each send/receive
- "Happens before" arcs must go from below to above tick
- Join ticks across space



Logical Clock Re-Order

- Straighten order of ticks
  - Note: Changes "order" of simultaneous p3 and q3



How to implement?

- Clock condition:
  - If event(A) happens before event(B),
    - C(A) < C(B) for all A,B
- IR1: Each process has local event count
- IR2: Tag messages with timestamps
  - Send with sender event count
  - Receive sets receiver clock = max(> incoming, local)
- Do something to establish *total* order from partial
  - e.g., concatenate unique PID to low bits of time
- Logical clocks are *very* common to let programmers reason in code. Many, many distributed systems...

Partial vs. Total Order

- Basic lamport clocks give a partial order
  - Many events happen "concurrently"
- But sometimes a total order is more convenient
  - A *consistent* total order
  - e.g., commit operations to a database
  - Or filesystem operations
  - Or RPCs, ...
- Different executions of deterministic logic may give different total orders, some logically incorrect (next lec) because of simultaneity errors

Distributed Mutex

- Not a very exciting example
  - who cares about granting in order they are *requested?*
  - but anyway... let's suspend disbelief, b/c other examples of this kind of algorithm really do matter
- Assumptions:
  - N messages sent as a single event (multicast)
    - All messages sent to all processes
  - Messages arrive reliably, in order sent
    - If not, add sequence #s, retransmit, buffer
- Fix messages between A&B to force 'happens b4'
- Queue order by sender timestamp, not receiver

The algo

- NOTE: Generalizes to arbitrary state machine!
- Pi sends Tm:Pi requests resource to all (+self)
- When Pj receives, places it on Q, send timestamped ack
- To release, Pi removes its own req from Q, sends timestamped Pi releases to all
- Pj receives release, removes Tm:Pi request from Q
- Pi gets resource if
  - Tm:Pi requests message first in Q by total ordering
  - Has received message >= Tm from everyone else (no outstanding messages from them that could contradict)

Has important kids

- Isis (Cornell, 80s)
  - Goal: Simplify programming for parallel machines/clusters
  - Provided both causally & totally ordered group communication
    - Translation: multicast and pub/sub
      - ISIS gave "exactly once" semantics to the *group*
      - All messages reach all receivers "at the same time"
    - Causal was 3x faster than total
    - But total is easier to program to
  - ISIS & derivatives: huge area of dist. sys research

ISIS Causal Order

- Each process keeps time vector of size N
- Start: VT[i] = 0
- When p sends message m, VT[p]++
- Message stamped with VTm (the VT of the sender)
- When p delivers message, p updates vec:
  - for i = 1..n: VTp[i] = max(VTp[i],VTm[i])
- VT1 <= VT2 iff for i=1..n:VT1[i] <= VT2[i]
- VT1 < VT2 iff VT1 <= VT2 && exists K s.t. VT1[k] < VT2[k]
- Causality: m1 -> m2 iff VT1 < VT2
- Can you deliver a message from q yet?
  - for i in 1..n
    - VTm[i] = VT[i] + 1 if i=q
    - VTm[i] <= VT[k] otherwise

ISIS derivatives

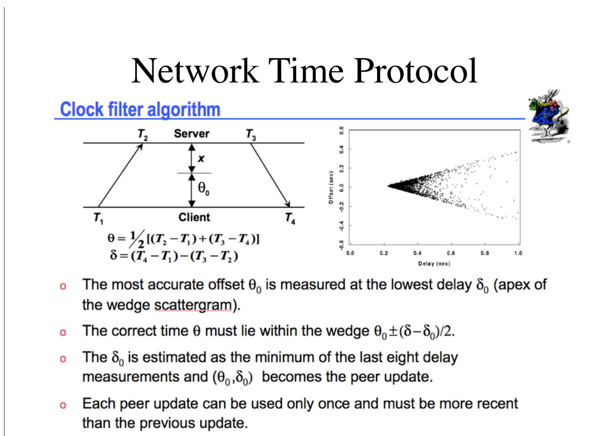
- "Version Vectors" for distributed filesystems (e.g., Coda), CVS, distributed shared memory, etc.
  - Same idea, but "clock" is changes to objects
- Later: Horus, Quicksilver
  - Improved group communication systems
  - Also Birman @ Cornell + MSR

Issues

- Failures: almost always physical timeout
  - Logical clocks have no notion of physical time
  - Failure tolerance is harder than program logic
- Covert channels can violate system causality
  - User interaction/input, filesystem access, etc.
  - e.g. phone call example in paper
- Integrating real clocks is tough...

Real Clocks

- Run at different rates
  - Your desktop probably gains or loses 1-30 seconds per day if not time-synched
  - And they drift over time, temp, etc.
- Synchronizing:
  - Use minimum delivery time
    - Lamport requires clock sync error < minimum transmission time (now microseconds!), but network clock sync gets milliseconds at best...
    - Not practical: so use NTP & live with covert channels
    - NTP: 10+ms on WAN, 100s usec on SAN, 100s nanosec using GPS. *But tough to get systems really set this well...*



- Accuracy bound: asymmetry in path
  - And things like unpredictable delays
    - Ethernet contention, interrupts, *missed* interrupts during high load (e.g., run "find" on disk), etc.

Evaluation

- Thought paper, but some big concepts:
  - Many computing events "logically" simultaneous
  - With causal links, partial ordering is key
  - Total orders easy to impose on partial order if needed
  - Broadcast-based group communication - important class of decentralized algorithms
  - Failure logic can't stay inside logical clock logic
  - Covert channels almost certainly will exist that defeat the logical clock logic
  - Real-time clock sync one option, but *hard*. If not hard, expensive!
  - Pretty decent clock sync based on message transmit time (NTP)