Time, Clocks, and the Ordering of Events in a Distributed System

Leslie Lamport
Massachusetts Computer Associates, Inc.

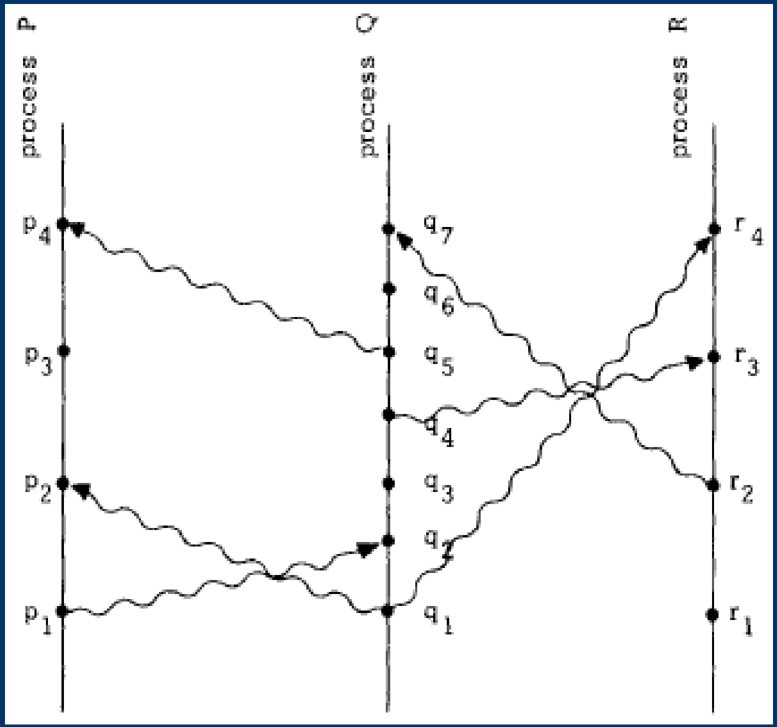
Presented J. Scott Miller

Event relationship framework

- A distributed system is thought of as a collection of processes on which a series of events occur
- Event a can be said to happen before b (a→b) under three conditions
  - If a and b are on the same process and a precedes b then a→b
  - If a is the sending of a message by one process and b is the receipt of that message by another process then a→b
  - If a→b and b→c then a→c
- Events for which a!→b and b!→a are said to be concurrent
- If a→b, there is a potential causal relationship between a and b

Event relationship framework (cont).

- Graphically, a→b means that we can traverse a space-time diagram from a to b
  - e.g. p1→r3
- Knowing that a→b means a possible causal relationship exists between those events



Logical clocks

- A logical clock assigns increasing numbers to each event, regardless of physical time
- If a→b then $C(a) < C(b)$
- Clock is maintained according to two rules
  - Each process increments C between any two events
  - Messages are timestamped with C at the sender, and the receiver much set its C to be greater than the timestamp

Total ordering

- An ordering of all system events can be achieved using the logical clock value and a process specific, unique value
- Event a on process i precedes event b on process j if either:
  - $C_i(a) < C_j(b)$
  - $C_i(a) = C_j(b)$ and $P_i < P_j$

Application: Resource Reservation

- Need to ensure exclusive access to certain system resources
  - e.g., atomic write access to a shared file
- Access should be granted in request order
- Processes should not be starved – the resource should eventually be released

Application: Resource Reservation (cont)

- To request a resource, a process sends a timestamped message to every other process to announce its intention
- Upon receiving a request, the process adds it to a private request queue
- When a resource is released, a process removes itself from its request queue and broadcasts a releases resource message – processes receiving this message do the same
- A process is granted the resource when it has a message requesting that resource in its queue ordered before any other request and has received a message from every other process timestamped later than the request message

Application: Resource Reservation (cont)

- Result is totally distributed
- Requires reliable, in-order messaging
- Each process retains state for all other processes
- Unresponsive or failing nodes need to be quickly discovered and their requests removed

Anomalous Behavior

- The total ordering introduced so far only to events and messages that occur within the system
- Messages transmitted outside of the system that result in an event might chronologically follow while logically preceding another event in the system
  - e.g., Phone call resulting in an event
- Two solutions
  - Include all possible events and messages in the system
  - Use a clock that can encapsulate outside effects

Physical clocks

- For a perfect physical clock, $dC(t)/dt = 1$
- κ is defined as the maximum clock error, $|dC(t)/dt - 1| < \kappa$
- ε is defined as the maximum difference between process clocks, for all i, j: $|Ci(t) - Cj(t)| < \varepsilon$
- μ is defined as the shortest allowable time between two events on remote processes, essentially the shortest possible transmission time
- Anomalous behavior is thus impossible if $\varepsilon/(1 - \kappa) <= \mu$
- Given a dense group of nodes, it is proved that this synchronization can be achieved with periodic messaging and a bounds on the unpredictable message delay