

Administrivia

- Your first programming assignment should be in today (I guess that means midnight)
 - The server should be running
 - I'll tell you (electronically) when to shut down
- Paper grades and comments will be delivered on Wednesday
 - I hope....

Techniques

- Knowing how and knowing that
 - Knowing how is technique
 - Knowing that is facts
- This is a class in knowing how
 - To know how, you need to learn some facts
 - But don't get confused
 - It's more than this class

Representing Time

- Three articles on logical clocks
 - Lamport (the original)
 - Babaoglu and Marzulo (many techniques)
 - Schneider (least formal)
- Different techniques illustrated

The Problem

- We prefer asynchronous systems
 - Everything true extends to synchronous
 - Real systems exhibit asynchronous traits
 - They provide an upper bound
- But we need order
 - Without causality, it is hard to think about things

Why Not sortOfsynchronous?

- Machines that are
 - Close in performance
- Networks with
 - Some bound on delay
- Synchronicity is binary

We've Heard

- Any synchrony is equivalent to real-time clocks
- Why?

Why Order?

- We want to know the state of the system
 - Is it deadlocked?
 - Is an invariant violated?
 - Do debits = credits?
- System state requires correlation
 - State is spread across systems

Let's Get Definitional

- A distributed system is
 - A set of processes, p_1, \dots, p_n
 - Connected by a network
- A distributed computation is
 - A set of events on those processors
 - Some of which are messages over the network

More Precisely

- For each processor, p_n ,
 - There is a set of events, $e^1_n, e^2_n, \dots, e^m_n$
 - Some events have the form $\text{send}(m)$ for message m
 - Some events have the form $\text{receive}(m)$ for message m
 - We will distinguish $\text{receive}()$ and $\text{deliver}()$ later
 - Events on a processor are ordered

A Distributed Computation

- For each processor, the events on that processor
 - Easy at the start and the finish
 - Hard elsewhere
- The state of a distributed computation
 - For each processor, the state it is in
 - How to find a consistent state

Histories:

Another Technique

- The history of a processor, p_n at m :
 - The set of events e^l_n, \dots, e^m_n
 - Implicitly, an order on those events
- The history of a computation:
 - A history for each process in the computation
- A history can be identified by its frontier

Space-time Diagrams

- A space-time diagram includes
 - The set of processors (one axis)
 - The sequence of events (the other axis)
 - Events for each processor
 - Arrows representing messages

Why All This Formalism

- Each formalism brings different tools
 - Sequence of events
 - Prefixes of infinite histories
 - Space-time diagrams
- Tools aid in thinking about the system
 - Use the tools that work for you
 - Know the others; they may work too

Ordering Everything

- Intuitive ordering
 - Events on a processor are easily ordered
 - Send(m) should be ordered before receive(m)
- More formally, we define \rightarrow as
 - If $e_n^m, e_n^{m'}$ are both simple events on processor p_n , then $e_n^m \rightarrow e_n^{m'}$ if $m < m'$
 - if e_n^m is an event of the form send(m) and $e_n^{m'}$ is an event of the form receive(m), then $e_n^m \rightarrow e_n^{m'}$
 - For all e, $\sim(e \rightarrow e)$

This Ordering

- Gives us consistency
 - A consistent state, S , is one such that if $e \in S$ and $e' \rightarrow e$ then $e' \in S$
- Is only a partial order
 - We can induce a total order
 - It may not correspond to reality

FIFO Delivery

- Goal: get FIFO delivery of messages
 - $\text{send}_i(m) \rightarrow \text{send}_i(m')$ then $\text{deliver}_j(m) \rightarrow \text{deliver}_j(m')$
- The approach (another technique)
 - Start with something you know works
 - Real-time clocks with bounded delay
 - Figure out how it works
 - Abstract

Start with Real-time Clocks

- Assume a real time clock, RC
 - The time at which e occurs is $RC(e)$
 - if $e \rightarrow e'$ then $RC(e) < RC(e')$
- If we have RC and a bound on delivery, δ
 - if $RC(m) < RC(m')$, then deliver(m) before m'
 - deliver(m) at $RC(m) + \delta$

Logical Clocks

- With logical clocks
 - The time at which e occurs is $LC(e)$
 - if $e \rightarrow e'$ then $LC(e) < LC(e')$
- If we have LC
 - if $LC(m) < LC(m')$, then deliver(m) before m'
 - deliver(m) when there is some m' from all processes such that $LC(m) < LC(m')$

So Logical Clocks

- Work
 - But they are a pain and
 - They don't scale, and
 - They aren't fault tolerant
- So next we will see another form of asynchronous clock

For Wednesday

- Read the RMI tutorial and “hello world” example
- Think about types in a distributed system