CS194: Clocks and Synchronization

Scott Shenker and Ion Stoica
Computer Science Division
Department of Electrical Engineering
and Computer Sciences
University of California, Berkeley
Berkeley, CA 94720-1776

1

Warning!

- Material is deceptively simple
- Looks obvious once you've thought about it
- But it took several years (and Lamport) to do the thinking
- And even the authors made errors….
- Just pay attention to the basic ideas, will get more detailed later in the course

2

When Did Time Begin?

- January 1, 1958
- In this lecture, when I refer to reference time, I mean universal coordinated time (UTC)

3

Why Do Clocks Matter?

- Correlate with outside world
  - Paper deadlines, earthquake analysis, etc.
  - Need true UTC
- Organize local process
  - File timestamps, etc.
  - Ordering, not UTC
- Coordinate between machines
  - Later: will use it in Kerboros, concurrency control, etc.
  - Rarely UTC, usually need *synchronized* ordering

4

Example: Make

- Make inspects files and compiles those that have changed since the last compilation.
- Assume file.c lives on machine 1, but file.o is produced on machine 2.
- If machine 2's clock is ahead of machine 1's, what might happen?

5

Make Example

	UTC	C1	C2
File compiled	10	5	15
File edited	15	10	20
Make initiated	20	15	25

File.c on 1: 10 File.o on 2: 15

Make does not recompile file

6

What Does Make Need?

- Machine needs to know if time of compilation is later than time of last edit.
- Ordering, not absolute time.
- Could be easily provided at the application level
  - Annotate file.o with timestamp of file.c

7

Assumptions

- Each machine has local clock
- No guarantee of accuracy, but never runs backwards
- Clocks on different machines will eventually differ substantially unless adjustments are made

8

Terminology

- Consider a clock with readings C(t), where t is UTC
- If C(t) = R x t + A then (different from book)
  A = offset
  R = skew
- If A or R change, that's drift
- Different clocks have different A, R

9

Adjusting Clocks

- Never make time go backwards!
  - Would mess up local orderings
- If you want to adjust a clock backwards, just slow it down for a bit

10

Aside #1: How Good are Clocks?

- Ordinary quartz clocks: 10^{-6} drift-seconds/second
- High-precision quartz clocks: 10^{-8}
- International Atomic Time (IAT): 10^{-13}
- GPS: 10^{-6} (why not just use GPS?)
- Computer clocks are lousy!

11

Clock Synchronization (Cristian)

- Client polls time server (which has external UTC source)
- Gets time from server
- How does it adjust this time?
  - Estimates travel time as 1/2 of RTT
  - Adds this to server time
- Problems? (major and minor)

12

Clock Synchronization (Berkeley)

- Time master polls clients (has no UTC)
- Gets time from each client, and averages
- Sends back message to each client with a recommended adjustment
- What advantages does this algorithm have?

13

Aside #2: Internet vs LAN

- Synchronizing at Internet scale is very different than synchronizing on a LAN
  - Delays more variable
  - Packet drops more likely

14

Network Time Protocol (NTP)

- Time service for the Internet
  - Synchronizes clients to UTC
- Primary servers connected to UTC source
- Secondary servers are synchronized to primary servers
- Clients synchronize with secondary servers

15

NTP (2)

- Reconfigurable hierarchy
  - Adapts to server failures, etc.
- Multiple modes of synchronization
  - Multicast (on LAN)
  - Server-based RPC
  - Symmetric (the fancy part!)
    - Pairs exchange messages
    - Attempt to model skew, offset
    - Signal processing to average out random delays
    - 10s of milliseconds over Internet

16

Aside #3: Sensornet Synchronization

- Leverages properties of broadcast:
  - Multiple receivers
  - Minimal propagation time
- Beacon sends out messages
  - Nodes receiving them compare timestamps
- Can extend to global synchronization
  - Neat mathematics….(clocks as rubber bands)
- On order of microseconds, not milliseconds
  - Why is this important?

17

Logical Clocks

- Who cares about time anyway?
- Ordering is usually enough
- What orderings do we need to preserve?

18

Lamport “Happens Before”

- A → B means A “happens before”
  - A and B are in same process, and B has a later timestamp
  - A is the sending of a message and B is the receipt
- Transitive relationship
  - A → B and B → C implies A → C
- If neither A → B nor B → A are true, then A and B are “concurrent” (not simultaneous)

19

Lamport Timestamps

- When message arrives, if process time is less than timestamp s, then jump process time to s+1
- Clock must tick once between every two events
- If A → B then must have L(A) < L(B)
- If L(A) < L(B), it does NOT follow that A → B
- How would this help make?

20

Make Example (revisited)

	UTC	C1	C2
Before compiled	10-	5-	15-
After compiled	10+	15+	15+
File edited	15	20	20
Make initiated	20	25	25

File.c on 1: 20   File.o on 2: 15

Make recompiles file

21

Ordering Noncausal Events

- Lamport timestamps don’t prevent two sites from processing events in different order
  - Lamport timestamps don’t unambiguously order events without a potential causal relationship
- In some cases (banking!) need everyone to process messages in the same order, even if there isn’t a causal order
- For that, we can use Totally Ordered Multicast

22

Totally Ordered Multicast

- Each message is broadcast to all other clients, with timestamp
- Each client broadcasts the ACK of that message
  - N^2 algorithm, not likely in the Internet….
- Only process head-of-queue (ordered by timestamp) when all ACKs received
- Why does this work?
- What happens when packets are reordered?

23

Totally Ordered Multicast (2)

- Don’t do event, then timestamp it.
- Declare your intention to do something, and timestamp that declaration
- Makes sure everyone hears that declaration
- All done in same order (not necessarily chronological!)

24

Aside #3: The Debate!

- There is a dispute about whether one needs highly synchronized primitives like totally ordered multicast
  - Recall, make problem handled by app-specific techniques
- Some contend that you should not embed heavyweight time ordering when most events don't need to be ordered
  - Only order important events using app-specific methods

25

Vector Timestamps

- L(A) < L(B) doesn't tell you that A came before B
- Only incorporates intrinsic causality, ignores any relationship with external clocks or external events
- Vector timestamps have the property that
  - V(A) < V(B) then A causally precedes B

26

Vector Timestamps (2)

- $V_I[I]$: number of events occurred in process I
- $V_I[J]$ = K: process I knows that K events have occurred at process J
- All messages carry vectors
- When J receives vector v, for each K it sets $V_J[K]$ = v[K] if it is larger than its current value

27

Vector Timestamps (3)

- If the vector associated event A is less than that associated with B, then A preceded B.
- This comparison is element by element
- Two vectors are "concurrent" if neither dominates the other
  - (1,5,1) vs (5, 1, 5)
- Why does this work?

28

Global State

- Global state is local state of each process, including any sent messages
  - Think of this as the sequence of events in each process
  - Useful for debugging, etc.
- If we had perfect synchronization, it would be easy to get global state at some time t
  - But don't have synchronization, so need to take snapshot with different times in different processes
- A consistent state is one in which no received messages haven't been sent
  - No causal relationships violated

29

Distributed Snapshot

- Initiating process records local state and sends out "marker" to its "neighbors"
- Whenever a process receives a marker:
  - Not recorded local state yet: records, then sends out marker
  - Already recorded local state: records all messages received after it recorded its own local state
- A process is done when it has received a marker along each channel; it then sends state to initiator

30

Termination

- Need distributed snapshot with no messages in flight
- Send “continue” message when finished with all channels, but not all have sent “done”
- Send “done” when all channels have sent “done” or when no other messages have been received since taking local state

31

Comments

- Few of these algorithms work at scale, with unreliable messages and flaky nodes
- What do we do in those cases?

32