Time
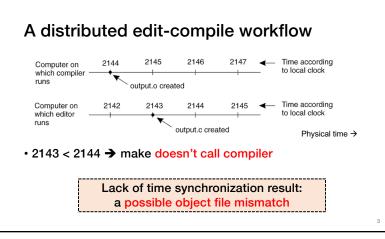
COS 418: Distributed Systems
Lecture 5

Mike Freedman

1

Today

1. The need for time synchronization

2. "Wall clock time" synchronization

3. Logical Time: Lamport Clocks

2

2

A distributed edit-compile workflow

• 2143 < 2144 ➔ make doesn't call compiler

Lack of time synchronization result:
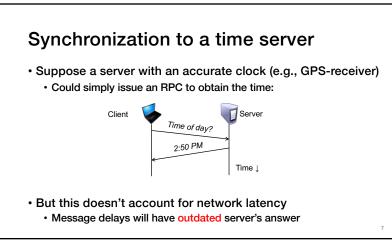a possible object file mismatch
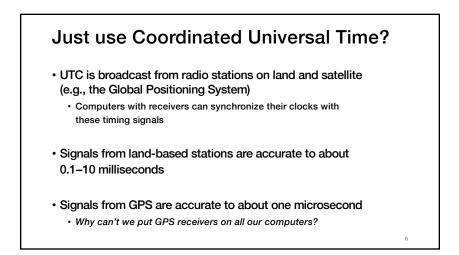
3

3

What makes time synchronization hard?

1. Quartz oscillator sensitive to temperature, age, vibration, radiation
   • Accuracy ~one part per million: one second of clock drift over 12 days

2. The internet is:
   • Asynchronous: arbitrary message delays
   • Best-effort: messages don't always arrive
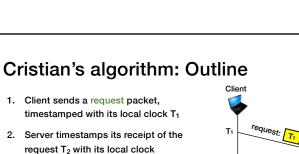
4

4

Today

1. The need for time synchronization

2. "Wall clock time" synchronization
   • Cristian's algorithm, NTP

3. Logical Time: Lamport clocks

5
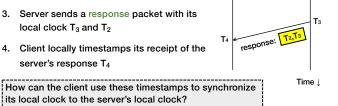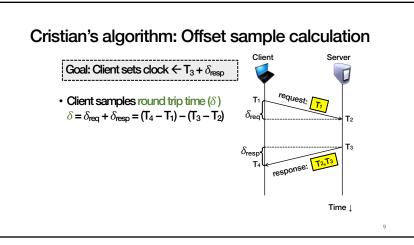
5

Just use Coordinated Universal Time?

• UTC is broadcast from radio stations on land and satellite (e.g., the Global Positioning System)
  • Computers with receivers can synchronize their clocks with these timing signals

• Signals from land-based stations are accurate to about 0.1–10 milliseconds

• Signals from GPS are accurate to about one microsecond
  • *Why can't we put GPS receivers on all our computers?*

6

6

Synchronization to a time server

• Suppose a server with an accurate clock (e.g., GPS-receiver)
  • Could simply issue an RPC to obtain the time:

Client   Server
Time of day?
2:50 PM
Time ↓

• But this doesn't account for network latency
  • Message delays will have outdated server's answer

7

7

Cristian's algorithm: Outline

Client   Server

1. Client sends a request packet, timestamped with its local clock T_1

2. Server timestamps its receipt of the request T_2 with its local clock

3. Server sends a response packet with its local clock T_3 and T_2

4. Client locally timestamps its receipt of the server's response T_4

T_1 request: T_1 T_2

T_3

T_4 response: T_2,T_3

Time ↓

How can the client use these timestamps to synchronize its local clock to the server's local clock?

8

8

2

Cristian's algorithm: Offset sample calculation

Goal: Client sets clock $\leftarrow T_3 + \delta_{resp}$

- Client samples round trip time (δ)
  $\delta = \delta_{req} + \delta_{resp} = (T_4 - T_1) - (T_3 - T_2)$

Client   Server

request: T_1
T_1
δ_{req} T_2

          T_3
δ_{resp}
T_4
response: T_2, T_3

Time ↓

9

Cristian's algorithm: Offset sample calculation

Goal: Client sets clock $\leftarrow T_3 + \delta_{resp}$

- Client samples round trip time (δ)
  $\delta = \delta_{req} + \delta_{resp} = (T_4 - T_1) - (T_3 - T_2)$

- But client knows δ, not δ_{resp}

  Assume: $\delta_{req} \approx \delta_{resp}$

Client sets clock $\leftarrow T_3 + \frac{1}{2}\delta$

Client   Server

request: T_1
T_1
δ_{req} T_2

          T_3
δ_{resp}
T_4
response: T_2, T_3

Time ↓

10

Clock synchronization: Take-away points

- Clocks on different systems will always behave differently
  - Disagreement between machines can result in undesirable behavior

- NTP clock synchronization
  - Rely on timestamps to estimate network delays
  - 100s μs–ms accuracy
  - Clocks never exactly synchronized

- Often inadequate for distributed systems
  - Often need to reason about the order of events
  - Might need precision on the order of ns

11

Today

1. The need for time synchronization

2. "Wall clock time" synchronization
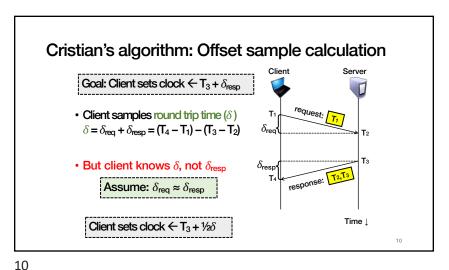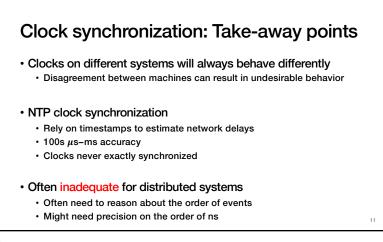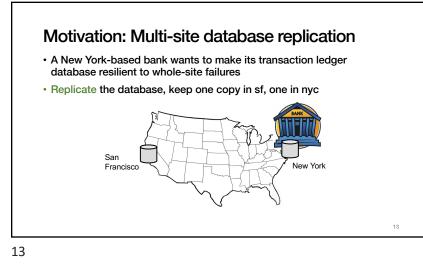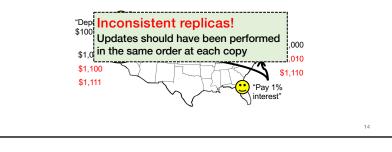   - Cristian's algorithm, NTP

3. Logical Time: Lamport clocks

12

3

Motivation: Multi-site database replication

- A New York-based bank wants to make its transaction ledger database resilient to whole-site failures

- Replicate the database, keep one copy in sf, one in nyc

San Francisco

New York

BANK

13

13

The consequences of concurrent updates

- Replicate the database, keep one copy in sf, one in nyc
  - Client sends reads to the nearest copy
  - Client sends update to both copies

"Dep $100

Inconsistent replicas!
Updates should have been performed in the same order at each copy

,000

$1,0

,010

$1,100

$1,110

$1,111

"Pay 1% interest"

14

14

RFC 677 (1975)
"The Maintenance of Duplicate Databases"

"To the extent that the communication paths can be made reliable, and the clocks used by the processes kept close to synchrony, the probability of seemingly strange behavior can be made very small. However, *the distributed nature of the system dictates that this probability can never be zero*."
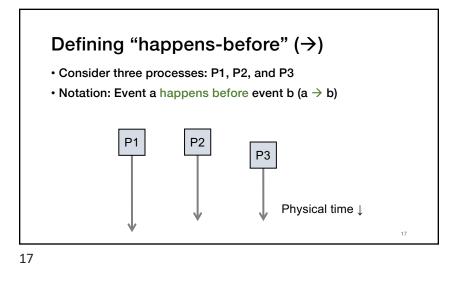
15

Idea: Logical clocks

- Landmark 1978 paper by Leslie Lamport
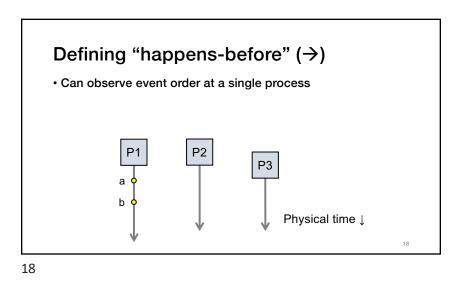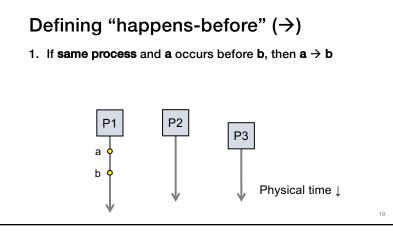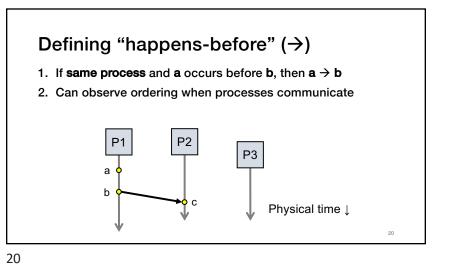
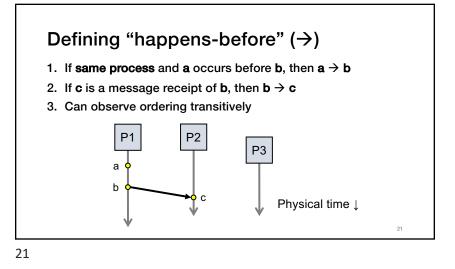- Insight: only the events themselves matter

Idea: Disregard the precise clock time

Instead, capture just a "happens before" relationship between a pair of events

16

16

Defining "happens-before" (→)

- Consider three processes: P1, P2, and P3
- Notation: Event a happens before event b (a → b)

P1   P2   P3   Physical time ↓

17

Defining "happens-before" (→)

- Can observe event order at a single process

P1   P2   P3

a
b

Physical time ↓

18

Defining "happens-before" (→)

1. If **same process** and **a** occurs before **b**, then **a → b**

P1   P2   P3

a
b

Physical time ↓

19

Defining "happens-before" (→)

1. If **same process** and **a** occurs before **b**, then **a → b**
2. Can observe ordering when processes communicate

P1   P2   P3

a
b   c

Physical time ↓

20

Defining "happens-before" (→)

1. If **same process** and **a** occurs before **b**, then **a** → **b**
2. If **c** is a message receipt of **b**, then **b** → **c**
3. Can observe ordering transitively

Physical time ↓

21

Defining "happens-before" (→)

1. If **same process** and **a** occurs before **b**, then **a** → **b**
2. If **c** is a message receipt of **b**, then **b** → **c**
3. If **a** → **b** and **b** → **c**, then **a** → **c**

Physical time ↓

22

Concurrent events

• Not all events are related by →
• a, d not related by → so concurrent, written as a ‖ d

Physical time ↓

23

Lamport clocks: Objective

• We seek a clock time C(a) for every event a

> Plan: Tag events with clock times; use clock times to make distributed system correct

• Clock condition: If a → b, then C(a) < C(b)

24

6

The Lamport Clock algorithm

- Each process P_i maintains a local clock C_i

1. Before executing an event, $C_i \leftarrow C_i + 1$

P1 $C_1=0$ P2 $C_2=0$ P3 $C_3=0$

a
b
c

Physical time ↓

25

25

The Lamport Clock algorithm

1. Before executing an event a, $C_i \leftarrow C_i + 1$:
   - Set event time $C(a) \leftarrow C_i$

P1 $C_1=1$ P2 $C_2=0$ P3 $C_3=0$

a C(a) = 1
b
c

Physical time ↓

26

26

The Lamport Clock algorithm

1. Before executing an event b, $C_i \leftarrow C_i + 1$:
   - Set event time $C(b) \leftarrow C_i$

P1 $C_1=2$ P2 $C_2=0$ P3 $C_3=0$

a C(a) = 1
b C(b) = 2
c

Physical time ↓

27

27

The Lamport Clock algorithm

1. Before executing an event b, $C_i \leftarrow C_i + 1$
2. Send the local clock in the message m

P1 $C_1=2$ P2 $C_2=0$ P3 $C_3=0$

a C(a) = 1
b C(b) = 2
c
C(m) = 2

Physical time ↓

28

28

7

The Lamport Clock algorithm

3. On process P_j receiving a message m:
   - Set C_j and receive event time $C(c) \leftarrow 1 + \max\{ C_j, C(m) \}$

P1 $C_1=2$
P2 $C_2=3$
P3 $C_3=0$

$C(a) = 1$
a
$C(b) = 2$
b
$C(c) = 3$
c
$C(m) = 2$

Physical time ↓

29

29

Lamport Timestamps: Ordering all events

- **Break ties** by appending the process number to each event:
  1. Process P_i timestamps event e with $C_i(e).i$
  2. $C(a).i < C(b).j$ when:
     - $C(a) < C(b)$, or $C(a) = C(b)$ and $i < j$

- Now, for any two events a and b, C(a) < C(b) or C(b) < C(a)
  - This is called a total ordering of events

30

30

Order all these events

P1 $C_1=0$
P2 $C_2=0$
P3 $C_3=0$
P4 $C_3=0$

a
b
d
f
h
c
e
i
g

Physical time ↓

31

31

Take-away points: Lamport clocks

- Can totally-order events in a distributed system: that's useful!
  - We saw an application of Lamport clocks for totally-ordered multicast

- But: while by construction, a → b implies C(a) < C(b),
  - The converse is not necessarily true:
    - C(a) < C(b) does not imply a → b (possibly, a ∥ b)

> **Can't** use Lamport timestamps to infer causal relationships between events

32

32

33