What is a distributed system?

- A collection of distinct processes that:
  - are spatially separated
  - communicate with each other by exchanging messages
  - have non-negligible communication delay
  - do not share fate
What is a distributed system?
Ordering events in a distributed system

What does it mean for an event to “happen before” another event?
What is a distributed system?

- A collection of distinct processes that:
  - are spatially separated
  - communicate with each other by exchanging messages
  - have non-negligible communication delay
  - do not share fate
  - have separate physical clocks

  (imperfect, unsynchronized)
Single machine

- A single clock
- Each event has a timestamp
- Compare timestamps to order events

Distributed system

- Each process has its own clock
- Each clock runs at a different speed
- Cannot directly compare clocks

an absolute temporal ordering is not what you want in a distributed system anyway

Leslie Lamport
Leslie Lamport

an absolute temporal ordering is not what you want in a distributed system anyway

Because temporal ordering is not observable. You cannot read two separate clocks simultaneously!

Why not?

High-level point: if a system is to meet a specification correctly, then that specification must be given in terms of events observable within the system
Ordering events without physical clocks

Modeling a process:

- A set of instantaneous events with an a priori total ordering
- Events can be local, sends, or receives.
Ordering events without physical clocks

“Happened-before” relation, denoted: \( \rightarrow \)

Part 1

- If \( a \) and \( b \) are events on the same process and \( a \) comes before \( b \), then \( a \rightarrow b \)

\[ \begin{align*}
\text{a} & \quad \text{b} \\
\cdot & \quad \cdot
\end{align*} \]
“Happened-before” relation, denoted: $\rightarrow$

Part 2

- If $a$ is the sending of a message by one process and $b$ is the receipt of the same message by another process, then $a \rightarrow b$
Ordering events without physical clocks

“Happened-before” relation, denoted: \( \rightarrow \)

Part 3

- If \( a \rightarrow b \) and \( b \rightarrow c \), then \( a \rightarrow c \)
Ordering events without physical clocks

Putting it all together

Diagram with points labeled a, b, c, d, e, f, g, h, i, j.
Ordering events without physical clocks

Can arrows go backwards?

Diagram: Nodes labeled a, b, c, d, e, f, g, h, i, j connected with arrows.
Can cycles be formed?

No, because the same event would happen at two different times.
Ordering events without physical clocks

Are all events related by $\rightarrow$ ?
A partial order

The set of events \( q \) such that \( q \rightarrow p \) are the events that could have influenced \( p \) in some way.

\[
\{a, b, e, f, h\}
\]
A partial order

If two events could not have influenced each other, it doesn’t matter when they happened relatively to each other.

\[ a \leftrightarrow b \leftrightarrow c \leftrightarrow d \]

\[ e \leftrightarrow f \leftrightarrow g \]

\[ h \leftrightarrow i \leftrightarrow j \]

\( h \) and \( d \) are concurrent: \( h \leftrightarrow d \), \( d \leftrightarrow h \).
Goal

- Generate a **total** order that is consistent with the happened-before partial order
  - E.g. $a \ b \ c \ d \ldots$
Lamport clocks

Define a function LC such that:

\[ p \rightarrow q \Rightarrow LC(p) < LC(q) \]

(the Clock condition)
Lamport clocks

Define a function LC such that:

\[ p \rightarrow q \Rightarrow LC(p) < LC(q) \]

(the Clock condition)

Implement LC by keeping a local LC_i at each process i
Lamport clocks

Single process

$\begin{array}{cccc}
 a & b & c & d \\
 1 & 2 & 3 & 4 \\
 6 & 37 & 1145 \\
\end{array}$
Lamport clocks

Across processes

\[ b \rightarrow h \Rightarrow LC(b) < LC(h) \]
\[ g \rightarrow h \Rightarrow LC(g) < LC(h) \]
Putting it all together
Is this correct?
Generating a total order

- Order messages by LC
- Ties are broken by unique process ID
Generating a total order

- Total order: $a, h, b, i, c, e, f, j, d, g$
Announcements

- No class for Lecture 1 next Thursday, 12/7

- Topics that you would like me to review?
  - Email me before class on Tuesday
  - I’ll review most popular requests during the last lecture
Course evaluations

- Please submit course evaluations!
- Incentive:
  - No. of bonus submissions on Project 4 = \( \text{max}(3, \ \text{floor}(\% \ of \ students \ who \ have \ submitted \ eval / 10)) \)
Distributed file systems

- Remote storage of data that appears local
- Examples:
  - NFS
  - Dropbox
  - Google Drive

- Benefits?
  - Share files among many users
  - Uniform view of file system across computers
Client-server design

- Copy of every file stored on server
- FS operations (read/write/etc.) are RPCs

Downsides?
- Network round-trip per low-level operation (Slow!)
- Server handles many operations (poor scaling)
Caching for performance

- Can cache file system data at client:
  - In memory (e.g., NFS)
  - On disk (e.g., AFS)

- Benefits of client-side caching?
  - Improves server scalability
  - Better latency and throughput
  - Reduces network traffic
  - Can improve availability (disconnected operation)
Client-side caching

- Migrate or replicate:
  - Migrate: Transfer sole copy from server to client
    » Simpler to implement
    » Concurrent reads lead to ping-ponging
  - Replicate: Create additional copy at client
    » Clients can read simultaneously
    » Must worry about inconsistent replicas
Pessimistic concurrency ctrl.

- How to prevent inconsistency?
  - Obtain “lock” before accessing data
  - Similar to reader-writer locks

- 3 states for each replica:
  - Invalid: no cached copy
  - Shared: may read cached copy
  - Exclusive: may read/write cached copy
State machine for cached copy

- Invalid
- Exclusive
- Shared

- This client writes the file
- Another client writes the file
- Another client reads the file
- Another client writes the file
- Another client reads the file
- This client writes the file
Invalidation protocol

Client

Read file
A

Write file
B

Server

Shared?
A

Shared, val=A

Exclusive?

Ack

Downgrade

Ack, val=B

Client

Read file

A

B

Shared?

Shared, val=A

Invalidate

Ack

Shared?

Ack

Val

B

Ack, val=B
Order of operations

- Necessary to wait for invalidation ACKs?

- Read file
  - Shared? → Shared, val=A
  - Exclusive? → Downgrade → Ack, val=B
  - Ack, val=B

- Write file
  - Shared?

- Server
  - Shared, val=A
  - Invalidate

- Client
  - Shared?
  - Ack, val=B
  - Shared, val=A

- Read file
Order of operations

- Necessary to wait for invalidation ACKs?

Client

Read file

A

Write file

B

Server

Shared?

Shared, val=A

Exclusive?

Invalidate

Client

Read file

A

Read file (A not B!)