Agenda

- Last time (D222 Thurs 5-6:15pm, D223 Fri 3-4:15pm)
  - Chapter 11 (time and global states)
- This time
  - More Chapter 11 Time and global states
- Next time (Thurs Apr 5)
  - Chpt 12: Coordination and Agreement

Before we start: Assignment #4 and #5

- We have selected Option 2
  - Assignment #4 (no book):
    - P2P out now, due Tues April 17 OR
    - "comparable" project proposal due Thurs Apr 5, due Tues Apr 17
  - Assignment #5 (only book): out April 5, April 10, due Tues April 24

Before we start: Schedule

<table>
<thead>
<tr>
<th>Sun</th>
<th>Tues</th>
<th>Thurs</th>
<th>Fri</th>
</tr>
</thead>
<tbody>
<tr>
<td>Coordination/agreement</td>
<td>3</td>
<td>4</td>
<td></td>
</tr>
<tr>
<td>Coordination/agreement</td>
<td>10</td>
<td>12</td>
<td></td>
</tr>
<tr>
<td>Transactions (13)</td>
<td>17</td>
<td>19</td>
<td></td>
</tr>
<tr>
<td>Google day due</td>
<td>24</td>
<td>26</td>
<td></td>
</tr>
<tr>
<td>Course wrap-up</td>
<td>1</td>
<td>3</td>
<td>11</td>
</tr>
<tr>
<td>Final – 2-5pm</td>
<td>8</td>
<td>10</td>
<td>11</td>
</tr>
</tbody>
</table>

Before we start: "comparable project"

- I think the P2P file system is very beneficial and fun, so I think you should do that
- However, you **ARE** allowed to pursue a different project
  - Which project? Let’s start a dialogue. Contact me.

Apr 24: Google Day!

- Possible papers
  - MapReduce: Simplified Data Processing on Large Clusters (OSDI 2004)
  - Google File System (SOSP 2003)
  - Bigtable: A Distributed Storage System for Structured Data (OSDI 2006)

- Interesting, but not explicitly covered:
  - How to design a Good API and Why it matters (OOPSLA’06)

Before we start: midterm answers

- l-b: stateless pro: fault-tolerance, con: msg size
- l-d: network transparency = access + location
  - From the client-perspective
- l-a: RPC runtime library includes [a] stubs, and [b] marshalling/demarshalling
  - Does not contain a “list” of methods
  - CAN contain lookup service, security service, etc.
- l-b: UDP-based RPC implementation, not just UDP-based (+2 points)
  - Stubs tend to default-select UDP or TCP. Client MIGHT notice if fails for various reasons (might be handled in stubs).
- l-c: RPC/RMI: client detects fault via time-out (+2 points)
Before we start: midterm answers

III-c: security mechanisms
- SSL and username/password
- Record transactions in database
- Use firewalls
- Patch software systems (e.g., Windows)
- Record employee modification of database

III-c: security policies
- User must use "strong" password
- User must modify password every 90 days
- Audit transactions every 30 days
- No entry point into system except through firewall
- Check for patches at least every 10 days
- Require manager to ok modification to "sensitive info"

IV-a: NFS security
- Group ID, user ID in cleartext (no "password" on wire)
- Client not particularly vulnerable (it's in the client-side kernel) more so than ANY app (e.g., rootkit)
- Server vulnerable via sniffing

IV-b: AFS vs. NFS: don’t give un-asked for details ("this is how AFS works…")
- Robustness: NFS is stateless but might be subject to scaling issues
- Potential for inconsistency: depends on particular use-case
  - E.g., two NFS clients concurrently editing. Maybe; MANY AFS clients "slowly" modifying, maybe not
- Network traffic generated: depends on particular use-case
  - AFS: large file with only read of small part: more traffic than necessary; NFS: opened for a long time with no mods: a lot of traffic to check for updates

Before we start: Midterm

Ambiguities in questions
- "UDP-based RPC implementation not just UDP": +2
- "RPC/RMI time-out mechanism": +2
- With these “additional” 4 points:
  - Max: 100
  - Min: 48
  - Average: 78
  - Median: 78

Chapter 11: Time and Global States: Roadmap

Problem definition: what would happen if there’s no attempt to synchronize clocks in a distributed system?
- Ground rules: Introduce a general model of computing (process, state, "happens before", HW clock, etc.)
- "approximate synchronization" of clocks
  - (internal) Synchronization in a synchronous system
  - Christian’s method (external)
  - Berkeley alg. (internal)
  - NTP (external)
  - Time
    Server B
    $T_{i-3}$
    $T_{i-2}$
    $T_{i-1}$
    $m$
    $m'$
    $m'$ carries $T_{i-3}$, $T_{i-2}$, $T_{i-1}$
  - we are here
- Logical time and logical clocks
  - Happened-before
  - Logical clocks
  - Global states

Network Time Protocol

Synchronization modes (cont’d):
- Procedure-call: (similar to Christian’s algorithm)
  - One server accepts requests from other computers
  - Replies by sending its timestamp (current clock reading)
  - This mode is suitable where higher accuracies are required (as compared to multicast)
Network Time Protocol (6)

\[
T_i = T_{i-1} + t' - o
\]

\[
D_i = t + t' = T_{i-2} - T_{i-3} + T_i - T_{i-1}
\]

\[
O_i = \frac{T_{i-2} - T_{i-3} + T_{i-1} - T_i}{2}
\]

\[
O = O_i + \frac{t' - t}{2}
\]

Time & Logical Clocks (1)

- Lamport: we cannot synchronize clocks perfectly; instead...
- Schemes and algorithms of distributed systems: importance of specifying if an event occurred before another or not
- Event \( e \) occurred before event \( f \) \( \iff \) occurrence time of event \( e \) is less than that of \( f \)
- \( \rightarrow \): causal ordering relation happened-before
- Within one process; \( P1: a, e, g, ... \)

\[ a \rightarrow e, e \rightarrow a \] concurrent events

Time & Logical Clocks (2)

- Relation \( \rightarrow \) can be applied to events belonging to different processes
- \( \text{Send}(\text{msg1}) \rightarrow \text{receive}(\text{msg1}) \)

Partial ordering of events

Time & Logical Clocks (3)

- Relation ‘happened before’ (\( \rightarrow \)) can be defined on a set of distributed systems’ events as follows:
  - HB1: If \exists\, \text{process } P_j: a \rightarrow b, \text{ then } a \rightarrow b
  - HB2: \forall\, m, \text{send}(m) \rightarrow \text{receive}(m)
  - HB3: If \( a, b \) and \( c \) are events such that \( a \rightarrow b \) and \( b \rightarrow c \), then \( a \rightarrow c \)
- Two events \( a \) and \( b \) are concurrent if \( a \rightarrow b \) and \( b \rightarrow a \)

Time & Logical Clocks (4)

- Two events \( a \) and \( b \) are concurrent if \( a \rightarrow b \) and \( b \rightarrow a \)

\[ a \rightarrow b \rightarrow c \rightarrow a \] concurrent events