Agenda

- Last time
  - Midterm discussion
  - More Chapter 11 (time and global states)
- This time
  - Finish Chapter 11 Time and global states
  - Chpt 12: Coordination and Agreement
- Next time (Tue Apr 10)
  - Chpt 12: Coordination and Agreement
- Remember: make-up class Thurs Apr 19/Fri Apr 20

Before we start: Schedule

<table>
<thead>
<tr>
<th>Sun</th>
<th>Tues</th>
<th>Thurs</th>
<th>Fri</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>Time/global (11)</td>
<td>3</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Coordination/agreement</td>
<td>4</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Assignment#5 Due</td>
<td>10</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Midterm Discussion</td>
<td>5</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Transactions (13)</td>
<td>12</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Transactions (14)</td>
<td>17</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Google day</td>
<td>24</td>
</tr>
<tr>
<td></td>
<td></td>
<td>PA5 due</td>
<td>3</td>
</tr>
<tr>
<td></td>
<td></td>
<td>No class – Marty out of town</td>
<td>26</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Final – 2-5pm</td>
<td>11</td>
</tr>
<tr>
<td>Sun</td>
<td>Tues</td>
<td>Thurs</td>
<td>Fri</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Course wrap-up</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Replication (15)</td>
<td>12</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Replication (15)</td>
<td>17</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Final – 2-5pm</td>
<td>11</td>
</tr>
</tbody>
</table>

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)

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)
- Logical time and logical clocks
  - Happened-before
  - Logical clocks ← we are here
  - Global states

From last time

- We can establish a partial ordering of events in a distributed system based on "happened before"
  - For any two events a and b, if we cannot establish that a → b or b → a, then a and b are said to be "concurrent"
- We can add a new other constraints and create a total ordering through the definition of a “logical clock”....

Time & Logical Clocks (8)

- Logical clock L: of process P, L: (monotonic) function that assigns a numeric value L(a) to each event a of P
- No particular relationship between logical and physical clocks
- Set of all logical clocks of all processes: can be represented by one function L
- Logical clock system is said to be correct, if it is consistent with the relation satisfies the condition:
  - For all events a and b, if a → b then L(a) < L(b)
- Implementing logical clocks (Lamport, 1978): use of timestamps
Process $P_i$: updates its logical clock and transmits its value within messages as follows:

1. **LC1**: $L_i$ is incremented before each event is issued at $P_i$.
2. **LC2**:
   - (a) When $P_i$ sends a message $m$, it piggybacks on $m$ a timestamp $T_m = L_i$.
   - (b) On receiving $(m, T_m)$, a process $P_j$ computes $L_j = \max(L_j, T_m)$ and then applies LC1 before timestamping the event receive($m$).

\[ a \rightarrow b \Leftrightarrow L(a) < L(b) \quad \text{(converse not true, } L(a) < L(b) \rightarrow a \rightarrow b) \]

Note: $L(b) > L(e)$ but $b \parallel e$.

**Total Ordering of Events**

- Using logical clocks to define a total order.
- Simple method: classify messages according to their timestamps.
- Problem: logical clock system may assign the same timestamp to different events.

Total relation $\triangleright$ for all events $a$ and $b$ occurred in two processes $P_i$ and $P_j$.

$$a \triangleright b \Leftrightarrow \begin{cases} 1) & L_i(a) > L_j(b) \text{, or} \\ 2) & L_i(a) = L_j(b) \text{ and } P_i < P_j \text{ in the conventional order of processes} \end{cases}$$

**Global States**

- If we had a single clock, we could snapshot all the processes at the same time and thus have our global state.
- System $\xi$: composed of $N$ processes $P_i$ ($i = 1, \ldots, N$).
  - Event $e$: either internal event in process, or sending a msg, or receiving a msg.
  - History $H_i = \langle e_i^1, e_i^2, e_i^3, \ldots \rangle$.
  - $S_i^k = \langle e_i^1, e_i^2, e_i^3, \ldots, e_i^k \rangle$: finite history.
  - $S_i^0 = \text{initial state of } P_i$.
  - $H = H_1 \cup H_2 \cup \ldots \cup H_N$: global history.
  - $S = (S_1, S_2, \ldots, S_N)$: global state of system $\xi$.

**Global States (2)**

- Objective: find out whether a particular property is true of a distributed system as it executes.
  - Distributed garbage collection: An object is considered as garbage if there are no longer references to it anywhere.
  - Distributed deadlock detection: detect if a collection of processes waits for another to send it a message.
  - Distributed termination detection: detect that a distributed algorithm has terminated.
  - Distributed debugging.

- A global consistent state corresponds to a consistent cut.

**Global States (3)**

- Cut: subset of the global history of a system’s execution.
  - Consistent cut $C$ is consistent for each event it contains, it also contains all the events that happened before that event.
  - Inconsistent cut $C$. 

**Global States (4)**

- Using of logical clocks to define a total order.
- Simple method: classify messages according to their timestamps.
- Problem: logical clock system may assign the same timestamp to different events.

**Global States (5)**

- Total relation $\triangleright$ for all events $a$ and $b$ occurred in two processes $P_i$ and $P_j$.

$$a \triangleright b \Leftrightarrow \begin{cases} 1) & L_i(a) > L_j(b) \text{, or} \\ 2) & L_i(a) = L_j(b) \text{ and } P_i < P_j \text{ in the conventional order of processes} \end{cases}$$

**Global States (6)**

- Objective: find out whether a particular property is true of a distributed system as it executes.
  - Distributed garbage collection: An object is considered as garbage if there are no longer references to it anywhere.
  - Distributed deadlock detection: detect if a collection of processes waits for another to send it a message.
  - Distributed termination detection: detect that a distributed algorithm has terminated.
  - Distributed debugging.

- A global consistent state corresponds to a consistent cut.

**Global States (7)**

- Inconsistent cut $C$. 

**Global States (8)**

- Using of logical clocks to define a total order.
- Simple method: classify messages according to their timestamps.
- Problem: logical clock system may assign the same timestamp to different events.

**Global States (9)**

- Objective: find out whether a particular property is true of a distributed system as it executes.
  - Distributed garbage collection: An object is considered as garbage if there are no longer references to it anywhere.
  - Distributed deadlock detection: detect if a collection of processes waits for another to send it a message.
  - Distributed termination detection: detect that a distributed algorithm has terminated.
  - Distributed debugging.

- A global consistent state corresponds to a consistent cut.
Global States

- **Snapshot algorithm (Chandy & Lamport 1985):** determines global states of distributed systems
- **Basic hypotheses:**
  - Neither channels nor processes fail. Reliable communication
  - Channels are unidirectional and provide FIFO-ordered message delivery
  - Graph of processes and channels is strongly connected (there is a path between any two processes)
  - Any process can initiate a global snapshot at any time
  - Processes: may continue their execution and send and receive normal messages while the snapshot takes place

**Snapshot algorithm (cont’d):**

**Marker receiving rule for process \( P_i \):**

On \( P_i \)'s receipt of a marker message over channel \( C \):

1. \( P_i \) records its process state now;
2. records the state of \( C \) as the empty set;
3. turns on recording of messages arriving over other incoming channels;

**Example:**

Initial state: \( P_2 \) has received order for 5 widgets @ $10/each; \( P_2 \) has not filled this order yet

Now let’s record the global state… (initiated by \( P_1 \), which acts as if it receives a marker over an imaginary link…)

**Global States (8)**

1. Global state \( S_0 \):
   - \( P_1 \): \(<\text{Order 10, $100}, \text{M}<$900, 0>\>
   - \( P_2 \): \(<\text{Order 10, $100}, \text{M} <$50, 2000>\>
2. Global state \( S_1 \):
   - \( P_1 \): \(<\text{Order 10, $100}, \text{M}<$900, 0>\>
   - \( P_2 \): \(<\text{Order 10, $100}, \text{M} <$50, 1995>\>
3. Global state \( S_2 \):
   - \( P_1 \): \(<\text{Order 10, $100}, \text{M}<$900, 5>\>
   - \( P_2 \): \(<\text{Order 10, $100}, \text{M} <$50, 1995>\>
4. Global state \( S_3 \):
   - \( P_1 \): \(<\text{Order 10, $100}, \text{M}<$900, 5>\>
   - \( P_2 \): \(<\text{Order 10, $100}, \text{M} <$50, 1995>\>

**Final state = \( \{ P_1, \langle \text{Order 10, $100}, \text{M} <$900, 5> \>; P_2, \langle \text{Order 10, $100}, \text{M} <$50, 1995> \} \)**

**NOTE:** system was never actually in this state!

Can you prove: [a] that this algorithm terminates?, [b] that the cut is consistent?