**Agenda**

- **Last time**
  - Chpt 12: Coordination and Agreement (mutual exclusion)
- **This time**
  - Chpt 12: Coordination and Agreement (election, consensus)
  - HW #5 out tomorrow (due last day of class, May 1)
- **Next time (Thurs Apr 19)**
  - Chpt 15: Replication
- **Next Next time (Thurs Apr 19/D222/Fri Apr 20 D223)**
  - Chpt 15: Replication
  - Note: we’re no longer planning to cover Transactions (no time!)

**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 and agreement</td>
<td>Replication (15)</td>
<td>Replication (15)</td>
<td></td>
</tr>
<tr>
<td>Assignment #4 due</td>
<td>17</td>
<td>19</td>
<td></td>
</tr>
<tr>
<td>Google day</td>
<td>No class – Marty out of town</td>
<td></td>
<td></td>
</tr>
<tr>
<td>24</td>
<td>26</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Course wrap-up</td>
<td>Assignment #5 due</td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>5</td>
<td></td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>10</td>
<td>Final – 2-5pm</td>
<td></td>
</tr>
<tr>
<td>11</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Apr 24: Google Day!**

- **Possible papers**
  - MapReduce: Simplified Data Processing on Large Clusters (OSDI 2004)
  - 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)

**Maekawa’s Voting Algorithm**

1. **Main steps of the algorithm:**
   - \[\text{state} \rightarrow \text{RELEASED};\]
   - voted = \text{FALSE};
   - For \( p \) in enter the critical section
     - state \( \rightarrow \text{WANTED}; \)
     - Multicast request to all processes in \( V_i \setminus \{ p \}; \)
     - Wait until (number of replies received = \( K - 1 \));
     - state \( \rightarrow \text{HELD}; \)
     - \( p \) enter the critical section only after collecting \( K-1 \) votes

2. **Optimal solution:** (minimize \( K \))
   - \( K = \sqrt{N} \) and \( M = K \)
   - Approximation:
     - Place all members in an \( \sqrt{N} \) by \( \sqrt{N} \) matrix and let \( V_i \) be the union of the row and column containing \( P_i \)
Maekawa’s Voting Algorithm (4)

- Main steps of the algorithm (cont’d):
  - \( \text{if (state = HELD OR voted = TRUE)} \)
    - \( \text{Then} \) queue request from \( p_i \) without replying;
    - \( \text{Else} \) reply immediately to \( p_i \);
    - \( \text{voted := TRUE; } \)
  - \( \text{On receipt of a request from } p_i \) at \( p_j \) (\( i \neq j \))
    - \( \text{If (state = HELD OR voted = TRUE)} \)
      - \( \text{Then} \) queue request from \( p_i \) without replying;
    - \( \text{Else} \) reply immediately to \( p_i \);
    - \( \text{voted := FALSE; } \)
  - \( \text{Multicast release to all processes } V_i - \{ p_i \}; \)

Maekawa’s Voting Algorithm (5)

- Main steps of the algorithm (cont’d):
  - \( \text{if (queue of requests is non-empty)} \)
    - \( \text{Then} \) remove head of queue, e.g., \( p_k \);
    - \( \text{send reply to } p_k; \)
    - \( \text{voted := TRUE; } \)
  - \( \text{Else voted := FALSE; } \)

Mutual Exclusion Algorithms Comparison

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>Number of messages</th>
<th>Problems</th>
</tr>
</thead>
<tbody>
<tr>
<td>Centralized</td>
<td>3</td>
<td>2</td>
</tr>
<tr>
<td>Virtual ring</td>
<td>1 to ( \infty )</td>
<td>0 to ( N-1 )</td>
</tr>
<tr>
<td>Logical clocks</td>
<td>( 2(N-1) )</td>
<td>( 2(N-1) )</td>
</tr>
<tr>
<td>Maekawa’s Alg.</td>
<td>( 3N )</td>
<td>( 2N )</td>
</tr>
</tbody>
</table>

Election Algorithms (1)

- Objective: Elect one process \( p_i \) from a group of processes \( p_1 \ldots p_n \)
- Utility: Elect a primary manager, a master process, a coordinator or a central server
- Each process \( p_i \) maintains the identity of the elected in the variable \( Elected \) (NIL if it isn’t defined yet)
- Properties to satisfy: \( \forall p_i \)
  - Safety: \( Elected = \text{NIL or } Elected = p_i \)
  - Liveness: \( p_i \) participates and sets \( Elected \neq \text{NIL}, \) or crashes

Election Algorithms (2)

- Ring-Based Election Algorithm
- Bully Algorithm
- Election Algorithms Comparison

Ring-Based Election Algorithm (1)

- \( \text{Message: } p_i \Rightarrow \)
  - Participant, := FALSE;
  - Elected, := NIL
  - \( \text{Send to } p_j \)
    - Participant, := TRUE;
    - Send the message <election, \( p_j \)> to its neighbor
- \( \text{Message: } p_i \Rightarrow \)
  - Participant, := FALSE;
  - If, \( p_i \neq p_j \)
    - Send the message <elected, \( p_j \)> to its neighbor
**Ring-Based Election Algorithm (2)**

Receipt of the election’s message <election, pi> at pj:

- If pi > pj
  - Then Send the message <election, pi> to its neighbor
  - Participantj := TRUE;
  - Else If pi < pj AND Participantj = FALSE
    - Then Send the message <election, pi> to its neighbor
    - Participantj := TRUE;
  
- Else If pi = pj
  - Then Electedj := TRUE;
  - Participantj := FALSE;
  - Send the message <elected, pj> to its neighbor

Hmmm... what happens if pi < pj AND Participantj = TRUE ??

---

**Ring-Based Election Algorithm (3)**

---

**Election Algorithm: Bully Algorithm**

- For process Pi to hold an election:
  - Pi sends an "election" message to all Pj, j>i
  - If no one responds, Pi declares itself the winner
  - If some Pj, j>i, responds Pi resigns.

- Winner sends a "coordinator" message to all other processes.
- Synchronous.

---

**Bully Algorithm cont.**

- p4 requests an election
- p5 and p6 reply cancel
- p5 and p6 hold elections
- p6 replies cancel and declares itself winner