

## 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!)**

---

CS451: Distributed Systems (Spring 2007)

## Before we start: Schedule

Sun	Tues	Thurs	Fri
	Coordination and agreement Assignment#4 due Assignment#5 out 17	Replication (15) Replication (15) 19	Replication (15)
	Google day 24	No class – Marty out of town 26	
	Course wrap-up Assignment#5 due 1		
			Final – 2-5pm 11
		8	10

---

CS451: Distributed Systems (Spring 2007)

## Apr 24: Google Day!

- **Possible papers**
  - The Anatomy of a Large-Scale Hypertextual Web Search Engine (Computer Networks, 1998)
  - Web Search for a Planet: The Google Cluster Architecture (IEEE Micro, 2003)
  - 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)

---

CS451: Distributed Systems (Spring 2007)

## Maekawa's Voting Algorithm (1)

- **Observation:** why bother those processes that don't care about the critical section at that particular time?
- "Processes need only obtain permission to enter from a subset of its peers, as long as the subsets used by any two processes overlap."
- Each process  $p_i$  maintain a *voting set*  $V_i$  ( $i=1, \dots, N$ ), where  $V_i \subseteq \{p_1, \dots, p_N\}$
- Sets  $V_i$ : chosen such that  $\forall i, j$ 
  - $p_i \in V_j$
  - $V_i \cap V_j \neq \emptyset$
  - $|V_i| = k$  (each process has the same size voting set)
- Each process  $p_i$  is contained in  $M$  of the voting sets  $V_i$

---

CS451: Distributed Systems (Spring 2007)

## Maekawa's Voting Algorithm (2)

- Optimal solution: (minimize  $K$ )
  - $K \sim \sqrt{N}$  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  $P_i$

---

CS451: Distributed Systems (Spring 2007)

## Maekawa's Voting Algorithm (3)

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

---

CS451: Distributed Systems (Spring 2007)

## Maekawa's Voting Algorithm (4)

- Main steps of the algorithm (cont'd):

On receipt of a request from  $p_i$  at  $p_j$ :

**If** (state = HELD OR voted = TRUE)

**Then** queue request from  $p_i$  without replying;

**Else** Reply immediately to  $p_i$ ;

voted := TRUE;

On receipt of a request from  $p_i$  at  $p_j$ :

state := RELEASED;

Multicast release to all processes  $V_i - \{p_j\}$ ;

CS451: Distributed Systems (Spring 2007)

## Maekawa's Voting Algorithm (5)

- Main steps of the algorithm (cont'd):

On receipt of a request from  $p_i$  at  $p_j$ :

**If** (queue of requests is non-empty)

**Then** remove head of queue, e.g.,  $p_k$ ;

send reply to  $p_k$ ;

voted := TRUE;

**Else** voted := FALSE;

CS451: Distributed Systems (Spring 2007)

## Mutual Exclusion Algorithms Comparison

Algorithm	Number of messages		Problems
	Enter/Exit	Before Enter	
Centralized	3	2	Crash of server
Virtual ring	1 to $\infty$	0 to N-1	Crash of a process Token lost Ordering non satisfied
Logical clocks	2(N-1)	2(N-1)	Crash of a process
Maekawa's Alg.	3√N	2√N	Crash of a process who votes

CS451: Distributed Systems (Spring 2007)

## Election Algorithms (1)

- Objective:** Elect one process  $p_i$  from a group of processes  $p_1 \dots p_N$
- 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  $Elected_i$  (NIL if it isn't defined yet)
- Properties to satisfy:**  $\forall p_i$ ,
  - Safety:**  $Elected_i = NIL$  or  $Elected = P$
  - Liveness:**  $p_i$  participates and sets  $Elected_i \neq NIL$ , or crashes

CS451: Distributed Systems (Spring 2007)

## Election Algorithms (2)

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

CS451: Distributed Systems (Spring 2007)

## Ring-Based Election Algorithm (1)

On receipt of a message:

Participant<sub>i</sub> := FALSE;

Elected<sub>i</sub> := NIL

On receipt of a message:

Participant<sub>i</sub> := TRUE;

Send the message <election,  $p_i$ > to its neighbor

On receipt of a message <elected,  $p_j$ > at  $p_i$ :

Participant<sub>i</sub> := FALSE;

**If**  $p_i \neq p_j$

**Then** Send the message <elected,  $p_j$ > to its neighbor

CS451: Distributed Systems (Spring 2007)

### Ring-Based Election Algorithm (2)

Example of the election message:  $\langle \text{election}, p_i \rangle$  at  $p_j$

If  $p_i > p_j$

Then Send the message  $\langle \text{election}, p_i \rangle$  to its neighbor  
Participant<sub>j</sub> := TRUE;

Else if  $p_i < p_j$  AND Participant<sub>j</sub> = FALSE

Then Send the message  $\langle \text{election}, p_j \rangle$  to its neighbor  
Participant<sub>j</sub> := TRUE;

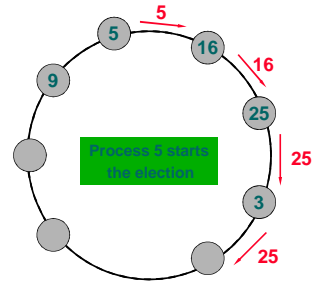
Else if  $p_i = p_j$

Then Elected<sub>j</sub> := TRUE;  
Participant<sub>j</sub> := FALSE;  
Send the message  $\langle \text{elected}, p_j \rangle$  to its neighbor

Hmmm... what happens if  $p_i < p_j$  AND Participant<sub>j</sub> = TRUE ??

CS451: Distributed Systems (Spring 2007)

### Ring-Based Election Algorithm (3)



CS451: Distributed Systems (Spring 2007)

### Election Algorithm: Bully Algorithm

#### For process $P_i$ to hold an election:

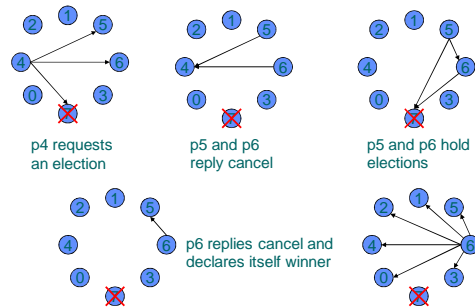
- $P_i$  sends an "election" message to all  $P_j, j > i$
- If no one responds,  $P_i$  declares itself the winner
- If some  $P_j, j > i$ , responds  $P_i$  resigns.

#### Winner sends a "coordinator" message to all other processes.

#### Synchronous.

CS451: Distributed Systems (Spring 2007)

### Bully Algorithm cont.



CS451: Distributed Systems (Spring 2007)