

Agenda

- **Last time**
 - Chpt 12: Coordination and Agreement (election)
- **This time**
 - HW#3 back
 - Chpt 12: Coordination and Agreement (group communication, consensus, Byzantine Generals)
 - HW #5 out (due last day of class, May 1) – **NO LATE SUBMISSIONS ACCEPTED!!!**
- **Today 5-6:15 D222/Tomorrow 3-4:15 D223**
 - Chpt 15: Replication
- **Next Tuesday: Google Day**
- **Next Thurs: no class**

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	3	
	8	10	Final – 2-5pm 11

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)

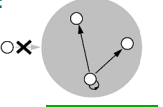
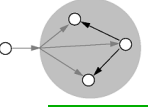
Group Communication (1)

- **Objective:** each of a group of processes must receive copies of the messages sent to the group
- **Group communication requires:**
 - Coordination
 - Agreement: on the set of messages that is received and on the delivery ordering
- We study multicast communication of processes whose membership is known (static groups)

CS451: Distributed Systems (Spring 2007)

Group Communication (2)

- **System:** contains a collection of processes, which can communicate **reliably** over **one-to-one** channels
- **Processes:** members of groups, may fail only by crashing
- **Groups:**

Group 1

Group 2

CS451: Distributed Systems (Spring 2007)

Group Communication (3)

- **Primitives:**
 - **multicast(g, m):** sends the message *m* to all members of group *g*
 - **deliver(m):** delivers the message *m* to the calling process
 - **sender(m):** unique identifier of the process that sent the message *m*
 - **group(m):** unique identifier of the group to which the message *m* was sent

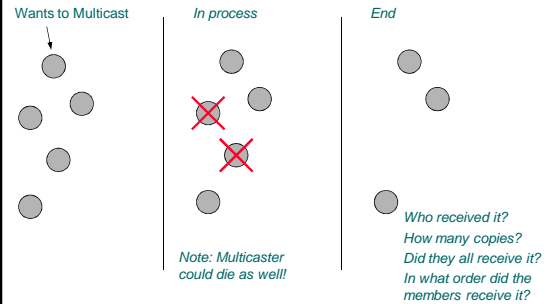
CS451: Distributed Systems (Spring 2007)

Group Communication (4)

- Basic Multicast
- Reliable Multicast
- Ordered Multicast (not covered in class)
 - E.g., if I multicast (g, m1) and then multicast (g, m2), will every process see m1 before m2?

CS451: Distributed Systems (Spring 2007)

Multicast Problem Definition



CS451: Distributed Systems (Spring 2007)

Basic Multicast

- Objective:** Guarantee that a correct process will eventually deliver the message as long as the multicaster does not crash
- Primitives:** B_multicast, B_deliver
- Implementation:** Use a reliable one-to-one communication
 - For each process $p \in g$, send(p, m);
 - B_deliver(m) to p
- Unreliable:** Acknowledgments may be dropped; note also not all processes get the message if the multicaster dies!

CS451: Distributed Systems (Spring 2007)

Reliable Multicast (1)

- Needed:** All correct processes must receive the message if ANY of them do (and only ONE copy)
- Properties to satisfy:**
 - Integrity:** A correct process P delivers the message m at most once
 - Validity:** If a correct process multicasts a message m, then it will eventually deliver m
 - Agreement:** If a correct process delivers the message m, then all other correct processes in group(m) will eventually deliver m (*atomicity*)
- Primitives:** R_multicast, R_deliver

CS451: Distributed Systems (Spring 2007)

Reliable Multicast (2)

Implementation using B-multicast:

```

msgReceived := {};
R-multicast(g, m) by p
  B-multicast(g, m); // p ∈ g
On receive(m) by p with q=sender(m) and g = group(m)
  If (m ∉ msgReceived)
  Then msgReceived := msgReceived ∪ {m};
  If (q ≠ p) Then B-multicast(g, m);
  R-deliver(m);
    
```

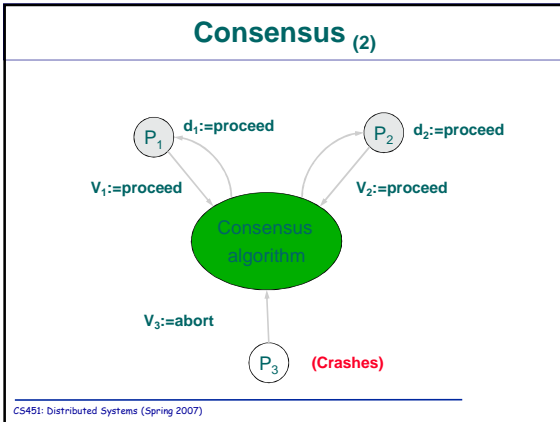
Pros/Cons?

CS451: Distributed Systems (Spring 2007)

Consensus (1)

- Objective:** processes must agree on a value after one or more of the processes has proposed what that value should be
- Why?** Should we land or takeoff?; Processes must agree on which processes to enter critical section;
- Hypotheses:** reliable communication, but processes may fail
- Consensus problem:**
 - Every process P_i begins in the *undecided* state
 - Every process proposes a value $V_i \in D$ ($i=1, \dots, N$)
 - Processes communicate with one another, exchanging values
 - Each process then sets the value of a decision variable d_i
 - Enters the state *decided*, in which it may no longer change d_i ($i=1, \dots, N$)

CS451: Distributed Systems (Spring 2007)



- ### Consensus (3)
- **Proprieties to satisfy:**
 - **Termination:** Eventually each correct process sets its decision variable
 - **Agreement:** the decision value of all correct processes is the same: P_i and P_j are correct $\rightarrow d_i = d_j$ ($i, j = 1, \dots, N$)
 - **Integrity:** If the correct processes all proposed the same value, then any correct process in the decided state has chosen that value
(e.g., if A and B propose "13", then stating that they agree to "15" would meet the "Agreement" requirement but intuitively not make sense)
- CS451: Distributed Systems (Spring 2007)

- ### Consensus (4)
- **Example: assume processes cannot fail**
 - Achieve consensus by:
 - Each process reliably multicasts its proposed value to the group
 - Each process waits until it has received all N proposed values
 - Each process then applies the *majority* function
 - Sets a value or NULL if no majority exists
 - Termination: through multicast
 - Agreement and Integrity: use and definition of *majority* and integrity of multicast
 - Note: arbitrary (byzantine) failures mess things up here.
- CS451: Distributed Systems (Spring 2007)

- ### Consensus (5)
- **Consensus in a synchronous system:**
 - Use of *basic multicast*
 - At most f processes may crash
 - $f+1$ rounds are necessary
 - Delay of one round is bounded by a timeout
- CS451: Distributed Systems (Spring 2007)

Consensus (6)

Algorithm for process $p_i \in g$; algorithm proceeds in $f+1$ rounds

On initialization
 $Values_i^0 := \{v_i\}; Values_i^0 = \{\};$

In round r ($1 \leq r \leq f+1$)
 $B\text{-multicast}(g, Values_i^r - Values_i^{r-1});$
 $Values_i^{r+1} := Values_i^r;$
 while (in round r)
 {
 On $B\text{-deliver}(V_j)$ from some p_j
 $Values_i^{r+1} := Values_i^{r+1} \cup V_j;$
 }
 }
 After $(f+1)$ rounds
 Assign $d_i = \text{minimum}(Values_i^{f+1});$

Correctness?

CS451: Distributed Systems (Spring 2007)

Correctness

- To show correctness \rightarrow show that every process has the same values at the end
- Assume the contrary..
 - What is the proof?

CS451: Distributed Systems (Spring 2007)

Byzantine Generals Problem

- Lamport *et. al.*, ACM Trans Prog Lang and Sys, Jul 82
- 3 or more generals are to agree to attack or to retreat
 - Requirements
 - All loyal generals decide upon the same plan of action
 - A small number of traitors cannot cause the loyal generals to adopt a bad plan
 - One, the commander, issues the order
 - The others (Lieutenant Generals) are to decide to attack or retreat (based on majority vote)
- One may be treacherous
 - Commander: tells one to attack, one to retreat
 - Lieutenants: tells one of his peers that the commander told him to attack and he tells another one that they are to retreat
- Overall issue: What percentage of liars can a decision-making algorithm tolerate and still correctly determine a consensus?
- Alg. Requirements
 - Termination: eventually each process sets its decision variable
 - Agreement: the decision value of all correct processes is the same
 - Integrity: if the commander is correct, then all correct processes decide on the value that the commander proposed

CS451: Distributed Systems (Spring 2007)

Byzantine Generals – Synchronous

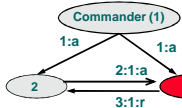
- Challenges
 - A FAULTY process can send any message with any value at any time
 - A FAULTY process can “forget” to send any message
- Up to f of the N processes may be faulty
- Note: Correct processes detect absence of message (why?), but they cannot conclude that the sender has crashed (why?)

CS451: Distributed Systems (Spring 2007)

Byzantine Generals: Consensus

- Byzantine agreement in a synchronous system:
 - Example : a system composed of three processes (must agree on “attack!” or “run!”)
- Impossibility of three processes:

Scenario 1: process 3 is faulty



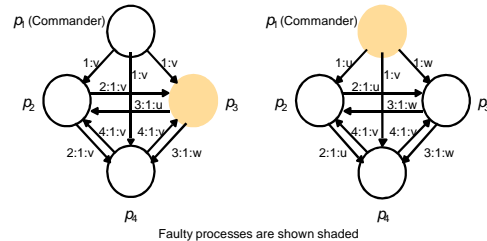
Scenario 2: Commander is faulty



Fisher (1985): no algorithm can guarantee to reach consensus in an asynchronous system, even with one process has a crash failure
 Pease (1980?): no solution exists if $N \leq 3f$

CS451: Distributed Systems (Spring 2007)

Four Byzantine Generals



Faulty processes are shown shaded

CS451: Distributed Systems (Spring 2007)