Announcements
- Midterm graded
- Milestone 2 coming up this Thursday

Midterm Prob 1a)
- Client and server on the same, different processes
  - Can you avoid marshaling/unmarshaling?
  - Can you pass arguments by reference?

Outline for Today
- Recap fault tolerance in distributed systems
- Atomic Multicast
- Distributed algorithms:
  - Election
  - Mutual Exclusion

Recap
- Fault Tolerance
  - Faults cause errors that lead to failure
    - Faults: transient/intermittent/permanent
    - Failure: crash-failure, omission failure, timing failure, response failure, arbitrary (or byzantine) failure
    - fail-stop vs. fail-silent
  - Redundancy can help tolerate faults by masking failure
    - Information ~, time ~, physical ~
    - Physical redundancy achieved through replication

Replicated State Machines
- Assume all replicas perform same set of operations in same order
- Atomic multicast
Results

- Reliable communication is critical
  - “Two Army Problem”
- K-fault tolerance
  - 1 client, replicated servers
    - Need k+1 servers if at most k can fail silent
    - Need 2k+1 if up to k byzantine failures and all respond (or fail stop)
    - Need 3k+1 if up to k byzantine failures and replies can be delayed

Distributed Agreement

- All communication synchronous (=delay bounded)
  - Need 3k+1 for k fault tolerance
    - “Byzantine Generals”
  - If communication is asynchronous (arbitrary delay)
    - Even 1-fault tolerance is impossible
    - Probabilistic Consensus: 3k+1 optimal
    - either liveness or safety guarantee must be sacrificed

Atomic Multicast

- Recall: for replication to make sense, all replicas must perform same set of operations and in same order
- Reduces to two subproblems in group communication
  - 1. Same set of messages: Virtual Synchrony
    - A message is either delivered to all processes in a group or to none
  - 2. Same order
    - (for both) Keyword: delivered (vs. received)

Message Receipt vs. Delivery

- Message is delivered to application
- Message is received by communication layer
- Message comes in from the network

Virtual Synchrony

- Observation: ensuring virtual synchrony requires agreement on group membership
- Group membership can change:
  - Processes may leave/crash
  - Processes may join/restart
- Idea: associate message m with current Group View (set of receivers)

Virtual Synchrony & Views

- Reliable multicast by multiple point-to-point messages
- P4 joins the group
- P5 crashes
- P5 reports
- P4 reports
Implementing VS in Isis

Atomic Multicast
= Virtual Synchrony + Total-order Delivery

Classification of virtual synchronous reliable multicasting:

<table>
<thead>
<tr>
<th>Message Ordering</th>
<th>None</th>
<th>FIFO Ordering</th>
<th>Causal Ordering</th>
</tr>
</thead>
<tbody>
<tr>
<td>Not total-ordered</td>
<td>Reliable Multicast</td>
<td>FIFO Multicast</td>
<td>Causal Multicast</td>
</tr>
<tr>
<td>Total-ordered</td>
<td>Atomic Multicast</td>
<td>FIFO Atomic Multicast</td>
<td>Causal Atomic Multicast</td>
</tr>
</tbody>
</table>

Summary Fault Tolerance
- Fault tolerance achieved via replication
- Feasibility of fault-tolerant replication strategies strongly depend on assumptions made and on agreement required
- Recall: Castro & Liskov
  - Practical algorithm for byzantine fault tolerance

Election Algorithms
- Need to select one “coordinator” process/node from several candidates
  - All processes should agree
- Use logical numbering of nodes
  - Coordinator is the one with highest number
- Bully algorithm
- Ring algorithm

Bully Algorithm (1)

Bully Algorithm (2)
Ring Algorithm

Mutual Exclusion
- Three criteria of critical sections
  - Mutual exclusion
    - “One at a time”
  - Progress
    - “May enter if empty”
  - Bounded Waiting
    - “Will enter eventually”
- Will look at three algorithms

Centralized ME w/ Coordinator

Distributed ME w/ Lamport Clocks

ME using Ring + Token
- Circulating Token
- Token holder may enter critical section

Comparison of ME algorithms

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>Messages per entry/exit</th>
<th>Delay before entry (in message times)</th>
<th>Problems</th>
</tr>
</thead>
<tbody>
<tr>
<td>Centralized</td>
<td>3</td>
<td>2</td>
<td>Coordinator crash</td>
</tr>
<tr>
<td>Distributed</td>
<td>2 ( (n - 1) )</td>
<td>2 ( (n - 1) )</td>
<td>Crash of any process</td>
</tr>
<tr>
<td>Token ring</td>
<td>1 to ( \infty )</td>
<td>0 to ( n - 1 )</td>
<td>Lost token, process crash</td>
</tr>
</tbody>
</table>

- Question: which is the most scalable? Fault-tolerant? Efficient?
Summary

• Distributed Synchronization
  – Election algorithms
  – Distributed Mutual Exclusion
    • Distributed \( \neq \) always better

• Thursday:
  – MapReduce (Google)
  – Coral (another P2P system)