Announcements

- Midterm next Tuesday
- Submit catchup work (if I asked you to) for milestone 1 by Nov 4

Recap

- Logical Clocks
  - Lamport Clocks: Scalar time, consistent
  - Lamport + Tiebreaker: Total order of events (but not related to happened-before)
  - Vector clocks: Strong consistency
    - Can tell if happened-before or concurrent

Outline for Today

- Introduction to fault tolerance in distributed systems
- Goal: Tolerate and recover from partial failure

Terminology

- Aspects of dependability:
  - Availability
  - Reliability
  - Safety
  - Security
  - Maintainability
- Types of faults:
  - Transient vs. intermittent vs. permanent

Failure Models

<table>
<thead>
<tr>
<th>Type of failure</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Crash failure</td>
<td>A server halts, but is working correctly until it halts</td>
</tr>
<tr>
<td></td>
<td>Fail-Stop: if this can be detected</td>
</tr>
<tr>
<td></td>
<td>Fail-Silent: if not (and others may wrongly believe in crash)</td>
</tr>
<tr>
<td>Omission failure</td>
<td>A server fails to respond to incoming requests</td>
</tr>
<tr>
<td>Receive omission</td>
<td>A server fails to receive incoming messages</td>
</tr>
<tr>
<td>Send omission</td>
<td>A server fails to send messages</td>
</tr>
<tr>
<td>Timing failure</td>
<td>A server’s response lies outside the specified time interval</td>
</tr>
<tr>
<td>Response failure</td>
<td>The server’s response is incorrect</td>
</tr>
<tr>
<td>Value failure</td>
<td>The value of the response is wrong</td>
</tr>
<tr>
<td>State transition failure</td>
<td>The server deviates from the correct flow of control</td>
</tr>
<tr>
<td>Arbitrary failure</td>
<td>A server may produce arbitrary responses at arbitrary times</td>
</tr>
</tbody>
</table>
Types of Redundancy

- Redundancy masks failures
- Information Redundancy
  - Example: Forward Error Correction (FEC)
- Time Redundancy
  - Example: Transactions
- Physical Redundancy
  - Example: Triple Modular Redundancy

Triple Modular Redundancy (TMR)

Types of Groups

Replication Protocols

- Primary-based protocols
  - hierarchical group
  - primary-backup protocols
  - NB: can mean either:
    - All updates are done at primary
    - Primary coordinates updates
- Replicated-write protocols
  - flat group
  - distributed consensus

K fault tolerance (in Repl. Write)

- A system is k-fault tolerant if it can tolerate faults in k components and still function
- Many results known, will present some of them
- Uses voting (single client here):
  - k+1 servers are k-fault tolerant for (up to k) silent failures
  - Byzantine failures: need 2k+1 servers. (Why?)

Practical Issues

- Failure independence
  - Assume that each node fails independently
  - When does this not apply? (How can we work around that?)
- How do we determine k anyway?
Revisit Assumptions

- K-fault tolerant result for replicated writes
  - (Silent-fail: k+1; Byzantine: 2k+1)
- Assumes synchrony:
  - Replies from nonfaulty replicas are received within known bounds
- Assumes all nonfaulty servers see same requests in the same order
  - Mechanism to ensure that: atomic multicast

Two-army Problem

- Unreliable communication channel:
  - Messages are can get lost
- Reliable processes:
  - Processes never crash
- Two-army problem:
  - Can two generals ("processes") communicating using messengers decide on a plan of attack? No.

Byzantine Generals Problems

- N generals, some of them are traitors
- Assumes communication is perfect, point-to-point: but traitors may lie
- Can loyal generals agree on plan of attack?
- To attack: must know troop strengths of all loyal generals

Byzantine Generals: 3+1

- Achieves agreement in synchronous systems: 3m+1 processes can tolerate up to m faulty processes
- Assumes communication is reliable + synchronous (think: "instantaneous")
- Assumes messages can be authenticated to have come from general G
  - Digital Signatures

Byzantine Generals: 2+1
Byzantine Agreement in Asynchronous Systems

- **Asynchronous**: Assumes reliable message transport, but with possibly unbounded delay
- Agreement is impossible with even one faulty process. Why?
  - Proof: Fischer/Lynch/Paterson 1985
  - Decision protocol must depend on single message:
    - Delayed: wait for it indefinitely
    - Or declare sender dead: may get wrong result

Rephrasing the problem

- "Probabilistic Consensus"
- Allow for protocols that may never terminate; but w/ probability zero
  - E.g., always terminates, but impossible to say when in the face of delay
- Result: Bracha/Toueg
  - Fail-stop: 2k+1
  - Byzantine: 3k+1
  - Optimal: both necessary + sufficient

Summary

- Fault tolerance achieved via replication
- Feasibility of fault-tolerant replication strategies strongly depend on assumptions made and on agreement required
- Thursday: Practical algorithm for byzantine fault tolerance