Last Class: Weak Consistency

- Eventual Consistency and epidemic protocols

- Implementing consistency techniques
  - Primary-based
  - Replicated writes-based
    - Quorum protocols

Gifford’s Quorum-Based Protocol

- Three examples of the voting algorithm:
  a) A correct choice of read and write set
  b) A choice that may lead to write-write conflicts
  c) A correct choice, known as ROWA (read one, write all)
Today: Fault Tolerance

- Basic concepts in fault tolerance
- Masking failure by redundancy
- Process resilience

Motivation

- Single machine systems
  - Failures are all or nothing
    - OS crash, disk failures
- Distributed systems: multiple independent nodes
  - Partial failures are also possible (some nodes fail)
- Question: Can we automatically recover from partial failures?
  - Important issue since probability of failure grows with number of independent components (nodes) in the systems
  - \( \text{Prob}(\text{failure}) = \text{Prob}(\text{Any one component fails}) = 1 - \text{P}(\text{no failure}) \)
A Perspective

• Computing systems are not very reliable
  – OS crashes frequently (Windows), buggy software, unreliable hardware, software/hardware incompatibilities
  – Until recently: computer users were “tech savvy”
    • Could depend on users to reboot, troubleshoot problems
  – Growing popularity of Internet/World Wide Web
    • “Novice” users
    • Need to build more reliable/dependable systems
  – Example: what is your TV (or car) broke down every day?
    • Users don’t want to “restart” TV or fix it (by opening it up)
• Need to make computing systems more reliable

Basic Concepts

• Need to build dependable systems
• Requirements for dependable systems
  – Availability: system should be available for use at any given time
    • 99.999 % availability (five 9s) => very small down times
  – Reliability: system should run continuously without failure
  – Safety: temporary failures should not result in a catastrophic failure
    • Example: computing systems controlling an airplane, nuclear reactor
  – Maintainability: a failed system should be easy to repair
Basic Concepts (contd)

- Fault tolerance: system should provide services despite faults
  - Transient faults
  - Intermitten faults
  - Permanent faults

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>Omission failure</td>
<td>A server fails to respond to incoming requests</td>
</tr>
<tr>
<td></td>
<td>Send omission</td>
</tr>
<tr>
<td></td>
<td>A server fails to receive incoming messages</td>
</tr>
<tr>
<td></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></td>
<td>Value failure</td>
</tr>
<tr>
<td></td>
<td>State transition failure</td>
</tr>
<tr>
<td></td>
<td>The value of the response is wrong</td>
</tr>
<tr>
<td></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>

- Different types of failures.
Failure Masking by Redundancy

• Triple modular redundancy.

Process Resilience

• Handling faulty processes: organize several processes into a group
  – All processes perform same computation
  – All messages are sent to all members of the group
  – Majority need to agree on results of a computation
  – Ideally want multiple, independent implementations of the application (to prevent identical bugs)
• Use process groups to organize such processes
Flat Groups versus Hierarchical Groups

Advantages and disadvantages?

Agreement in Faulty Systems

- How should processes agree on results of a computation?
- *K-fault tolerant*: system can survive k faults and yet function
- Assume processes fail silently
  - Need \((k+1)\) redundancy to tolerant k faults
- *Byzantine failures*: processes run even if sick
  - Produce erroneous, random or malicious replies
    - Byzantine failures are most difficult to deal with
  - Need \(?\) Redundancy to handle Byzantine faults
Byzantine Faults

- Simplified scenario: two perfect processes with unreliable channel
  - Need to reach agreement on a 1 bit message
- Two army problem: Two armies waiting to attack
  - Each army coordinates with a messenger
  - Messenger can be captured by the hostile army
  - Can generals reach agreement?
  - Property: Two perfect process can never reach agreement in presence of unreliable channel
- Byzantine generals problem: Can N generals reach agreement with a perfect channel?
  - M generals out of N may be traitors

Byzantine Generals Problem

- Recursive algorithm by Lamport
- The Byzantine generals problem for 3 loyal generals and 1 traitor.
  a) The generals announce their troop strengths (in units of 1 kilosoldiers).
  b) The vectors that each general assembles based on (a)
  c) The vectors that each general receives in step 3.
Byzantine Generals Problem Example

- The same as in previous slide, except now with 2 loyal generals and one traitor.
- Property: With \( m \) faulty processes, agreement is possible only if \( 2m + 1 \) processes function correctly [Lamport 82]
  - Need more than two-thirds processes to function correctly