CSE 486/586 Distributed Systems
Failure Detectors

Steve Ko
Computer Sciences and Engineering
University at Buffalo

Recap
• Three most important things
  – Read the documentation.
  – Do it; write your code.
  – Learn how to debug.
• Android programming model
  – Event-driven
  – Hidden main() calls appropriate callbacks depending on events from outside.
• AsyncTask and Threading
  – Typically Android has one main thread: UI thread.
  – You can create new threads using AsyncTask and Java Thread.

Today’s Question
• How do we handle failures?
  – Cannot answer this fully (yet!)
• You’ll learn new terminologies, definitions, etc.
• Let’s start with some new definitions.

Two Different System Models
• Synchronous Distributed System
  • Each message is received within bounded time
  • Each step in a process takes lb < time < ub
    • (Each local clock’s drift has a known bound)
    • Examples: Multiprocessor systems
• Asynchronous Distributed System
  • No bounds on message transmission delays
  • No bounds on process execution
  • (The drift of a clock is arbitrary)
  • Examples: Internet, wireless networks, datacenters, most real systems
• These are used to reason about how protocols would behave, e.g., in formal proofs.

Failure Model
• What is a failure?
  • We’ll consider: process omission failure
    • A process disappears.
    • Permanently: crash-stop (fail-stop) – a process halts and does not execute any further operations
    • Temporarily: crash-recovery – a process halts, but then recovers (reboots) after a while
• We will focus on crash-stop failures
  • They are easy to detect in synchronous systems
  • Not so easy in asynchronous systems

Why, What, and How
• Why design a failure detector?
  – First step to failure handling
• What do we want from a failure detector?
  – No miss (completeness)
  – No mistake (accuracy)
• How do we design one?
What is a Failure Detector?

Crash-stop failure
(p is a failed process)

needs to know about p's failure
(p is a non-faulty process or alive process)

There are two styles of failure detectors:

• p queries p once every T time units
  • If p does not respond within another T time units of being sent the ping, p detects/declares p as failed

• p maintains a sequence number
  • p sends p a heartbeat with incremented seq. number after every T time units

The waiting time 'T' can be parameterized.

In a Synchronous System

• The Ping-Ack and Heartbeat failure detectors are always correct. For example,
  – Ping-Ack: set waiting time 'T' to be > round-trip time upper bound
  – Heartbeat: set waiting time '3T' to be > round-trip time upper bound

• The following property is guaranteed:
  – If a process p fails, then p will detect its failure as long as p itself is alive
  – Its next ack/heartbeat will not be received (within the timeout), and thus p will detect p as having failed
Failure Detector Properties

- What do you mean a failure detector is “correct”?
- **Completeness** = every process failure is eventually detected (no misses)
- **Accuracy** = every detected failure corresponds to a crashed process (no mistakes)
- What is a protocol that is 100% complete?
- What is a protocol that is 100% accurate?
- Completeness and Accuracy
  - Can both be guaranteed 100% in a synchronous distributed system (with reliable message delivery in bounded time)
  - Can never be guaranteed simultaneously in an asynchronous distributed system
    - Why?

Completeness and Accuracy in Asynchronous Systems

- Impossible because of arbitrary message delays, message losses
  - If a heartbeat/ack is dropped (or several are dropped) from pj, then pj will be mistakenly detected as failed => inaccurate detection
  - How large would the T waiting period in ping-ack or 3T waiting period in heartbeating, need to be to obtain 100% accuracy?
  - In asynchronous systems, delay/losses on a network link are impossible to distinguish from a faulty process
- Heartbeating – satisfies completeness but not accuracy (why?)
- Ping-Ack – satisfies completeness but not accuracy (why?)

Completeness or Accuracy? (in Asynchronous System)

- Most failure detector implementations are willing to tolerate some inaccuracy, but require 100% completeness.
- Plenty of distributed apps designed assuming 100% completeness, e.g., p2p systems
  - “Err on the side of caution”.
  - Processes not “stuck” waiting for other processes
- But it’s ok to mistakenly detect once in a while since
  - the victim process need only rejoin as a new process
- Both Heartbeating and Ping-Ack provide
  - Probabilistic accuracy (for a process detected as failed, with some probability close to 1.0 (but not equal), it is true that it has actually crashed).

Failure Detection in a Distributed System

- That was for one process pj being detected and one process pi detecting failures
- Let’s extend it to an entire distributed system
- Difference from original failure detection is
  - We want failure detection of not merely one process (pj), but all processes in system

CSE 486/586 Administrivia

- PA2 will be out probably tonight.
- Please use Piazza; all announcements will go there.
  - If you want an invite, let me know.
- Please come to my office during the office hours!
  - Give feedback about the class, ask questions, etc.
Centralized Heartbeat

Ring Heartbeat

All-to-All Heartbeat

Efficiency of Failure Detector: Metrics

• **Bandwidth**: the number of messages sent in the system during steady state (no failures)
  – Small is good

• **Detection Time**
  – Time between a process crash and its detection
  – Small is good

• **Scalability**: Given the bandwidth and the detection properties, can you scale to a 1000 or million nodes?
  – Large is good

• **Accuracy**
  – Large is good (lower inaccuracy is good)

Accuracy Metrics

• **False Detection Rate**: Average number of failures detected per second, when there are in fact no failures

• Fraction of failure detections that are false

• Tradeoff: If you increase the T waiting period in ping-ack or 3*T waiting period in heartbeating what happens to:
  –Detection Time?
  – False positive rate?
  – Where would you set these waiting periods?

Other Types of Failures

• Let’s discuss the other types of failures

• Failure detectors exist for them too (but we won’t discuss those)
Processes and Channels

![Diagram of Processes and Channels]

Other Failure Types

• **Communication omission failures**
  - Send-omission: loss of messages between the sending process and the outgoing message buffer (both inclusive)
  » What might cause this?
  - Channel omission: loss of message in the communication channel
  » What might cause this?
  - Receive-omission: loss of messages between the incoming message buffer and the receiving process (both inclusive)
  » What might cause this?

Other Failure Types

• **Arbitrary failures**
  - Arbitrary process failure: arbitrarily omits intended processing steps or takes unintended processing steps.
  - Arbitrary channel failures: messages may be corrupted, duplicated, delivered out of order, incur extremely large delays, or non-existent messages may be delivered.
  - Above two are Byzantine failures, e.g., due to hackers, man-in-the-middle attacks, viruses, worms, etc.
  - A variety of Byzantine fault-tolerant protocols have been designed in literature!

Omission and Arbitrary Failures

<table>
<thead>
<tr>
<th>Class of failure</th>
<th>Affects</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Fail-stop</td>
<td>Process</td>
<td>Process halts and remains halted. Other processes may detect this state.</td>
</tr>
<tr>
<td>Omission</td>
<td>Channel</td>
<td>A message inserted in an outgoing message buffer never arrives at the other end’s incoming message buffer.</td>
</tr>
<tr>
<td>Send-omission</td>
<td>Process</td>
<td>A process completes send but the message is not put in its outgoing message buffer.</td>
</tr>
<tr>
<td>Receive-omission</td>
<td>Process</td>
<td>A message is put in a process’s incoming message buffer, but that process does not receive it.</td>
</tr>
<tr>
<td>Arbitrary (Byzantine)</td>
<td>Channel</td>
<td>Process or channel exhibits arbitrary behaviour: it may send/transmit arbitrary messages at arbitrary times, commit omissions; a process may stop or take an incorrect step.</td>
</tr>
</tbody>
</table>

Summary

• Failure detectors are required in distributed systems to keep system running in spite of process crashes
• Properties – completeness & accuracy, together unachievable in asynchronous systems but achievable in synchronous systems
  - Most apps require 100% completeness, but can tolerate inaccuracy
• 2 failure detector algorithms - heartbeating and ping
• Distributed FD through heartbeating: centralized, ring, all-to-all
• Metrics: bandwidth, detection time, scale, accuracy
• Other types of failures
• Next: the notion of time in distributed systems

Acknowledgements

• These slides contain material developed and copyrighted by Indranil Gupta at UIUC.