Failure Detection

DISTRIBUTED SYSTEMS – CS425/ECE428 – SPRING 2021
Announcements

Exams
- Dates: TBA
- Process: online, open-book
- ES-Based (Exam Scheduler™): proctored by course staff over Zoom, fewer restrictions than CBTF

Lecture back-channel
- Use the CampusWire #lecture room for questions during the lecture

MP0
- Released tomorrow
- Partner search thread on CampusWire
Key Properties

Multiple computers
- Concurrent execution
- Independent failures
- Autonomous administrators
- Heterogeneous capacities, properties
- Large numbers (scalability)

Networked communication
- Asynchronous execution
- Unreliable delivery
- Insecure medium

Common goal
- Consistency – can discuss whole-system properties
- Transparency – can use the system without knowing details
Objectives

How do we detect failures?

Models
  ◦ Failures
  ◦ Networks

Properties
  ◦ Guarantees
  ◦ Metrics

Techniques
Failure Model

What is a failure?

Process omission failure
  ◦ Crash-stop (fail-stop) – a process halts and does not execute any further operations
  ◦ Crash-recovery – a process halts, but then recovers after a while

Encompasses both hardware and software failures
What’s a failure detector?

$p_i$  $p_j$
What’s a failure detector?

Crash-stop failure
(p_j is a \textit{failed} process)

\[ p_i \]
What’s a failure detector?

needs to know about $p_j$’s failure
($p_i$ is a non-faulty process
or alive process)

Crash-stop failure
($p_j$ is a failed process)
I. Ping-Ack Protocol

$p_i$ sends a ping every $T$ time units

$p_j$ replies as soon as it receives a ping

When to consider process $p_j$ failed?

- $\Delta_1$ after sending ping and no ack
- $\Delta_1 = \text{maximum round-trip time (why?)}$
II. Heartbeating Protocol

$p_j$ sends a heartbeat every $T$ time units

When to consider $p_j$ as failed?
- $T + \Delta_2$ after receiving last heartbeat
- $\Delta_2 = \text{maximum one-way delay}$
  - $[- \text{minimum one-way delay}]$
Two Different System Models

Synchronous Distributed System
- Each message is received (successfully) within bounded time
- Each step in a process takes $lb < \text{time} < ub$
- (Each local clock’s drift has a known bound)

Asynchronous Distributed System
- No bounds on message transmission delays
- No bounds on process execution
- (The drift of a clock is arbitrary)

Which is more realistic?
- Synchronous: Multiprocessor systems
- Asynchronous: Internet, wireless networks, datacenters, most real systems
Correctness properties

Completeness = every process failure is eventually detected (no misses)

Accuracy = every detected failure corresponds to a crashed process (no mistakes)

In a synchronous system, ping-ack and heartbeat are complete and accurate

In an asynchronous system, impossible to have both completeness and accuracy
  ◦ Cannot reliably distinguish failure and unbounded delay
Completeness v. Accuracy

Is Ping-ack complete, accurate, or neither?
- Complete: a failure *will* result in missing ack

Is Heartbeat complete, accurate, or neither?
- Complete *(mostly)*

Can we have an accurate (but incomplete) algorithm in asynchronous systems?
- Never report failure
Accuracy

Systems are usually designed around complete but inaccurate detectors

- Advantage: cannot be stuck forever waiting for failed process
- Disadvantage: have to recover somehow if made a mistake

Why bother with failure detectors at all?

- “Always report failure” is also complete but inaccurate
- Probabilistic accuracy: reported failure => actual failure with high (but < 1) probability