

## CSE 486/586 Distributed Systems Failure Detectors

Steve Ko  
Computer Sciences and Engineering  
University at Buffalo

CSE 486/586

## Recap

- Best Practices

CSE 486/586

2

## 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.
- One of the two fundamental challenges in distributed systems
  - Failure
  - Ordering (with concurrency)

CSE 486/586

3

## Two Different System Models

- **Synchronous Distributed System**
  - Each message is received within bounded time
  - Each step in a process takes  $l_b < \text{time} < u_b$
  - (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.

CSE 486/586

4

## 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

CSE 486/586

5

## 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?

CSE 486/586

6

## What is a Failure Detector?



CSE 486/586

7

## What is a Failure Detector?



Crash-stop failure  
( $p_j$  is a failed process)

CSE 486/586

8

## What is 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)

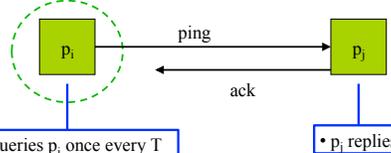


There are two styles of failure detectors

CSE 486/586

9

## I. Ping-Ack Protocol



•  $p_i$  queries  $p_j$  once every  $T$  time units

• If  $p_j$  does not respond within another  $T$  time units of being sent the ping,  $p_i$  detects/declares  $p_j$  as failed

•  $p_j$  replies

If  $p_j$  fails, then within  $T$  time units,  $p_i$  will send it a ping message.  $p_i$  will time out within another  $T$  time units.

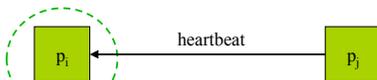
Worst case Detection time =  $2T$

The waiting time ' $T$ ' can be parameterized.

CSE 486/586

10

## II. Heartbeating Protocol



• If  $p_i$  has not received a new heartbeat for the past, say  $3T$  time units, since it received the last heartbeat, then  $p_i$  detects  $p_j$  as failed

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

If  $T \gg$  round trip time of messages, then worst case detection time  $\sim 3T$  (why?)

The ' $3$ ' can be changed to any positive number since it is a parameter

CSE 486/586

11

## In a Synchronous System

- The Ping-Ack and Heartbeat failure detectors are **always correct**. For example (there could be other ways),
  - Ping-Ack: set waiting time ' $T$ ' to be  $>$  round-trip time upper bound
  - Heartbeat: set waiting time ' $3 \cdot T$ ' to be  $>$  round-trip time upper bound
- The following property is guaranteed:
  - If a process  $p_j$  fails, then  $p_i$  will detect its failure as long as  $p_i$  itself is alive
  - Its next ack/heartbeat will not be received (within the timeout), and thus  $p_i$  will detect  $p_j$  as having failed

CSE 486/586

12

## 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?

CSE 486/586

13

## Completeness and Accuracy in Asynchronous Systems

- Impossible because of **arbitrary message delays**
  - If a heartbeat/ack is dropped (or several are dropped) from  $p_j$ , then  $p_j$  will be mistakenly detected as failed => inaccurate detection
  - How large would the  $T$  waiting period in ping-ack or  $3 \cdot T$  waiting period in heartbeating, need to be to obtain 100% accuracy?
  - In asynchronous systems, **delays 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?)
- **Point: You can't design a perfect failure detector!**
  - You need to think about what metrics are important.

CSE 486/586

14

## 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).

CSE 486/586

15

## Failure Detection in a Distributed System

- That was for one process  $p_j$  being detected and one process  $p_i$  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 ( $p_j$ ), but **all** processes in system

CSE 486/586

16

## CSE 486/586 Administrivia

- PA2A due in two weeks (Fri, 2/19)
- 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.

CSE 486/586

17

## Failure Detection in a Distributed System

- That was for one process  $p_j$  being detected and one process  $p_i$  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 ( $p_j$ ), but **all** processes in system
-  Any idea?
  - Why
  - What
  - How

CSE 486/586

18

### 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)

CSE 486/586

19

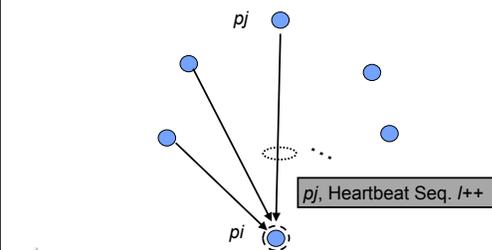
### Accuracy Metrics

- **False Detection Rate:** Average number of failures detected per second, when there are in fact no failures
- Fraction of failure detections that are false
- Tradeoffs: If you increase the T waiting period in ping-ack or  $3 \cdot T$  waiting period in heartbeating what happens to:
  - Detection Time?
  - False positive rate?
  - Where would you set these waiting periods?

CSE 486/586

20

### Centralized Heartbeat

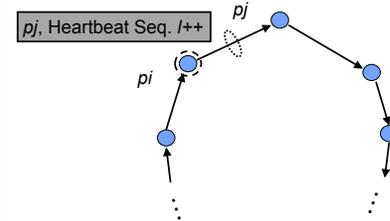


*Downside?*

CSE 486/586

21

### Ring Heartbeat

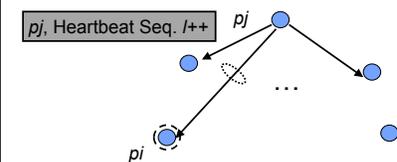


*Downside?*

CSE 486/586

22

### All-to-All Heartbeat



*Advantage: Everyone is able to keep track of everyone*

*Downside?*

CSE 486/586

23

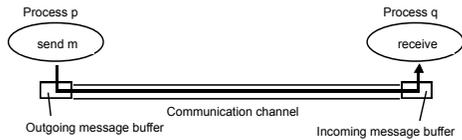
### Other Types of Failures

- Let's discuss the other types of failures
- Failure detectors exist for them too (but we won't discuss those)

CSE 486/586

24

## Processes and Channels



CSE 486/586

25

## 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?

CSE 486/586

26

## 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!

CSE 486/586

27

## Omission and Arbitrary Failures

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

CSE 486/586

28

## 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

CSE 486/586

29

## Acknowledgements

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

CSE 486/586

30