Failure and Failure Detection

CSE 486: Distributed Systems

Ethan Blanton

Department of Computer Science and Engineering

University at Buffalo

Failures

Detecting failures is important in distributed systems.

It is also difficult:

- Asynchronous systems may have unpredictable delay
- Failed components may be "somewhere else"
- Failed components may not be under your control

Failure detection is necessary for failure recovery.

Failure Types

Failures come in many types.

- A process stops responding.
- Messages are lost.
- A process behaves incorrectly.
- Messages are corrupted.

These can be separated into crash, omission and arbitrary failures [2].

A process may also simply be too slow or too fast.

Crash Failures

The simplest type of crash failure is fail-stop [4].

This means process halts and never takes another action.

Processes may also:

- Pause: A process halts and later continues where it left off
- Have amnesia: A process restarts from an initial state.
- Have partial amnesia: A process remembers some state, but other state is reset to an initial state.

These are often more difficult to handle than fail-stop.

Omission Failures

Omission failures have two types:

- Send Omission: A process fails to send a message
- Receive Omission: A process fails to receive (or process) a message

Note that these may be indistinguishable from loss!

Arbitrary Failures

Other types of failures may include:

- Response: A process behaves or replies incorrectly
- Byzantine: A process displays different failures to different observers

Arbitrary (and particularly Byzantine) failures are difficult.

Failure Detection Model

We will focus on fail-stop conditions.

We wish to identify a halted process.

The halted process will never run again.

Note that fail-stop can sometimes be emulated.
(A process that experiences another failure can just stop!)

A non-failed process is correct.

Properties of Detection

Failure detectors have two related properties:

Completeness: A failed process will always be identified.

Accuracy: A correct process will never be considered failed.

The weakest useful [1] failure detector satisfies:

1. There is a time after which every failed process is always suspected by some correct process.
2. There is a time after which some correct process is never suspected by any correct process.

Completeness and Accuracy

Completeness: A failed process will always be identified.

Accuracy: A correct process will never be considered failed.

What is a 100% complete detector?

Completeness and Accuracy

Completeness: A failed process will always be identified.

Accuracy: A correct process will never be considered failed.

What is a 100% complete detector?

What is a 100% accurate detector?

Completeness and Accuracy

Completeness: A failed process will always be identified.

Accuracy: A correct process will never be considered failed.

What is a 100% complete detector?

What is a 100% accurate detector?

Are they useful?

Messages

Processes in our system only interact via messages.

This means that the only indication of failure is lack of messages.

How can we reliably detect a lack of messages?

Messages

Processes in our system only interact via messages.

This means that the only indication of failure is lack of messages.

How can we reliably detect a lack of messages?

Ensure that we can expect messages!

Heartbeats

A common method of failure detection is heartbeats.

A process periodically sends a message.

While messages are being received, the process is not failed.

If no message is received for some time, the process is failed.

In a synchronous system, this is complete and accurate.

Asynchrony

In an asynchronous system, this is can be complete.

Asynchrony

In an asynchronous system, this is can be complete.

Proof by contradiction:

- P sends n heartbeat messages before failing, p ms apart.
- Q expects a message every $\leq 2p$ ms.

Assume that Q believes P is alive after $2np + \epsilon$ ms.

Asynchrony

In an asynchronous system, this is can be complete.

Proof by contradiction:

- P sends n heartbeat messages before failing, p ms apart.
- Q expects a message every $\leq 2p$ ms.

Assume that Q believes P is alive after $2np + \epsilon$ ms.

P would have had to have sent $n + 1$ messages.

Asynchrony

In an asynchronous system, this is can be complete.

Proof by contradiction:

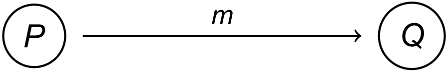
- P sends n heartbeat messages before failing, p ms apart.
- Q expects a message every $\leq 2p$ ms.

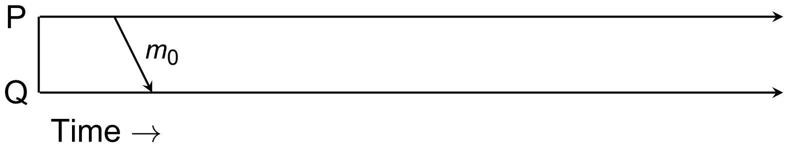
Assume that Q believes P is alive after $2np + \epsilon$ ms.

P would have had to have sent $n + 1$ messages.

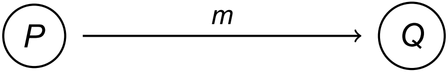
It cannot be accurate, however.

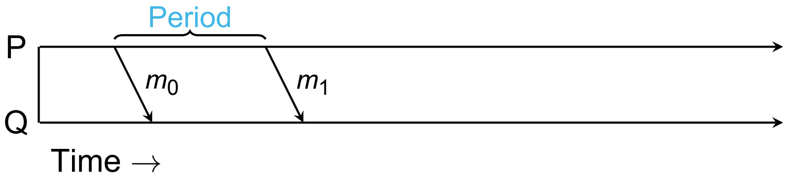
Simple Detector



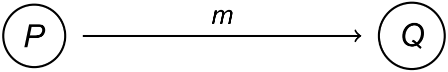


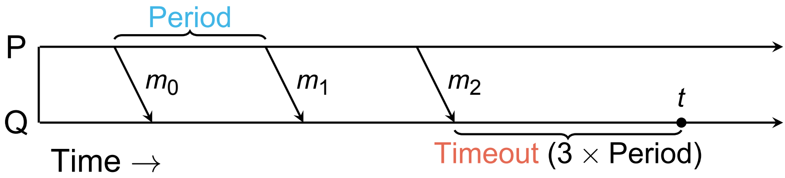
Simple Detector





Simple Detector





Q determines that P has failed at time t.

Loss vs. Failure

What does message loss do to this system?

What does that say about the timeout length?

Cost of Failure Detection

Heartbeat has cost tradeoffs:

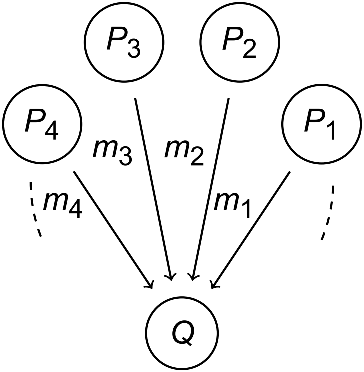
- Messages sent (network traffic)
- Time to completeness
- Accuracy rate

Higher message rates and short timeouts improve completeness.

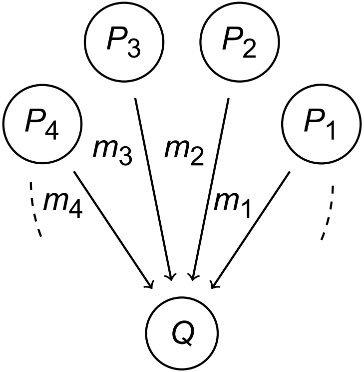
Longer timeouts improve accuracy.

Lower message rates reduce the cost of communication.

Central Detector

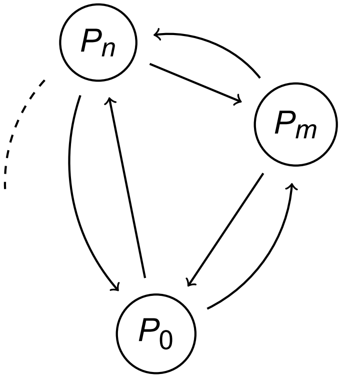


Central Detector



What are the advantages and disadvantages of this method?

All-Pairs Detector



What are the advantages and disadvantages of this method?

Summary

- Failure detection is important for distributed systems.
- There are many possible types of failures:
  - Crash
  - Omission
  - Response
  - Byzantine
- Crash failures and message loss can be confused.
- Completeness and accuracy are measures of failure detector goodness.
- Asynchronous system failure detectors cannot be both.

Next Time …

- Time!

References I

Required Readings

[3] Ajay D. Kshemkalyani and Mukesh Singhal. *Distributed Computing: Principles, Algorithms, and Systems*. Chapter 15: 15.1, 15.2, 15.7. Cambridge University Press, 2008. ISBN: 978-0-521-18984-2.

Optional Readings

[1] Tushar Deepak Chandra, Vassos Hadzilacos, and Sam Toueg. *The Weakest Failure Detector for Solving Consensus*. Tech. rep. 94-1426. Cornell Computer Science, May 1994. URL: https://ecommons.cornell.edu/bitstream/handle/1813/6208/94-1426.pdf.

References II

[2] Flaviu Cristian. "Understanding Fault-Tolerant Distributed Systems". In: *Communications of the ACM* 34 (2 Feb. 1991), pp. 57–78. URL: https://dl.acm.org/doi/abs/10.1145/102792.102801.

[4] Richard D. Schlichting and Fred B. Schneider. "Fail-Stop Processors: An Approach to Designing Fault-Tolerant Computing Systems". In: *ACM Transactions on Computing Systems* 1.3 (Aug. 1983), pp. 222–238. URL: https://www.cs.cornell.edu/fbs/publications/Fail_Stop.pdf.