Detecting Failures in Distributed System with the FALCON Spy Network

Joshua B. Leners, Hao Wu, Wei-Lun Hung, Marcos K. Aguilera, Michael Walfish

Present by Yuchen Wang
November 14, 2018
Agenda

• Motivation
• Existing Solutions on failure detection
• FALCON spy network - High Level Design
• Spy Implementation
• Evaluation on FALCON
• Discussions
Motivations

• Problem

• A good distributed system (in data center!) should tolerate crash failures.

• First, Detect, and then, recover. Existing solutions for detection have shortcomings.
Failure Detection: Conventional Wisdom

**Fast:** Detection time should be less than 1 second.

**Reliable:** Report the process UP only if it is operational (DOWN only if it fails).

**Undisruptive:** Avoid to kill operational processes.

**CHOOSE TWO**
(The Asynchronous communication environment)
Existing Solutions

- **End-to-end Timeout:**
  - Slow and blunt (Tens of seconds at least)
  - GFS & NFS: 60s, Chubby: 12s

- **STONITH based detectors (Shot The Other Node In The Head)**
  - First of all, ruthless
  - More importantly, causes lots of disruptions.

- **Special Hardware, for a Real-time Synchronous System**
  - Too expensive to datacenter, not viable.

Why not use Paxos? It’s very Simple.
Problem Recap

- Building a good failure detector is hard.

- Fast, reliable, undisruptive, you can only achieve two out of three, fundamentally because of the asynchronous environment.
Really impossible to have ALL THREE?

• High level Intuitions:

• **Intuition 1**: Many crash failures can be observed by looking at the right layer.

• **Intuition 2**: The existing solutions are too specific, let’s take each other’s advantages, and try to combine them.
Basic Idea of Falcon

- **FALCON** (Fast And Lethal Component Observation Network)
  - Use a chain of “spies” to monitor different **layers** of the system
  - Layers from high to low: Applications-> Operating System -> Virtual Machine Monitor -> Network Switch
  - Use inside information to monitor (**fast**)
  - The Lower-level spies also monitor higher-level spy.

- Kill the layer when uncertain (**guarantees reliability**), but try to kill as few as possible (**minimize disruption**)

- Use end to end timeout as backstop. Triggered only for exception.
High Level Architecture

- A client library on client side
- Chained Spies for each layer
- Spies **actively** send the state to client library.
Look into the spy

• Spy for Layer L: Inspector on L, enforcer on L - 1

• As long as Layer L - 1 is operational, the spy for L is responsive.
An Implementation Example of Spies: Application Spy
The Spy for the Application!
The Spy for the Application!
The Spy for the Application!
The Spy for the Application!
The Spy for the Application!
The Spy for the Application!

Client Side

Server Side

Application Layer

OS layer

Process Table

App Enforcer

App Inspector
f()

Client Library

UP

Measure CPU time, NOT real-time -> minimize false suspicions
Server Side

Application

App Inspector

f()

App Enforcer

Application Layer

OS layer

Process Table

Client Side

Client Library

Down

The Spy for the Application!
App Enforcer

Application

App Inspector

f()

Application Layer

OS layer

Process Table

App Enforcer

Client Side

Client Library

One Spy for the Application layer!
Server Side

Client Side

One Spy for the Application layer!
Server Side

Client Side

Application

App Inspector

f()

Application Layer

OS layer

Process Table

One Spy for the Application layer!
App Enforcer
Application
App Inspector
f()
One Spy for the Application layer!
Client Side

Application Layer
OS layer

Process Table

One Spy for the Application layer!

Client Library
One Spy for the Application layer!
An Implementation Example of Spies: OS Spy, VMM Spy

• Similar idea as application spy
An Implementation Example of Spies: Network Spy

- Across the network (lives on both client side and remote server).
- Doesn’t check for failures of switch.
- FALCON Blocks, when the switch is unresponsive (e.g network partition).
End-to-end Timeout

• Provided by the client library. Controlled by the user.

• Catch unexpected Exception, like a “catch” block in c++

• Kill the layers from up to down. (e.g first try to kill application layer, then OS...).
Application Restart

• The application may lose part of its state in a crash, so the client should know.

• Use a generation number to distinguish between each instance.

• Lower layer restarts -> higher layer should also updates.
Evaluation

• Fast:
  • 99% detection time within hundred ms,
  • 10 - 100 times faster than existing solutions
  • Reduce unavailability by 6x (Zookeepers)

• Little disruption:
  • Kill the smallest component

• Reliable:
  • Enables primary-backup and make it reliable.

• Inexpensive:
  • Single digit CPU overhead for each layer
  • Easy to integrate with application (tens or hundreds lines of code)
  • 50% less overhead than Paxos, 21% less code (for primary backup).
Evaluation: Median Detection Time

The diagram shows the median detection time in seconds for various types of events. The x-axis represents different types of events such as app crash, app layer-down report, kernel hang, kernel stack overflow, kernel panic, VMM error/guest exit, and host down. The y-axis represents the median detection time in seconds, with lower times being better. The error bars indicate the variability of the detection times.
Recap & Discussions

• FALCON Spy Network:
  • Chained spy structure and monitoring.
  • End-to-end timeout to catch unexpected condition.
  • Can almost achieve all three properties: fast, reliable, undisruptive.

• Conventional Wisdom IS NOT WRONG
  • Using of Inside information & chained structure largely leverage the performance
  • Real system are NOT ALWAYS asynchronous.

• ONLY Crash failures!
Questions?