Predicting and Validating Behavior in Distributed Systems

Javier Álvarez Cid-Fuentes
Supervised by Claudia Szabo and Katrina Falkner
The University of Adelaide
javier.alvarez@adelaide.edu.au

4th NICTA Software Systems Summer School
February 9th, 2016
Introduction

- Complexity in distributed systems leads to unexpected behaviors [1]

- Examples [1]:
  - Livelock
  - Deadlock
  - Unwanted synchronization

- Lead to: → Economic losses
  - Performance loss
  - Failures
Real World Cases

• Oct 2012 Amazon's 2 hours outage [8]:
  - May cost $66,000 per minute [9]
  - “our monitoring failed to alarm on this memory leak”

• Jan 2014 Google's 30 min outage caused by a software bug in an internal system [10]:
  - Could cost $10,000 per minute [11]
  - “[...] adding additional targeted monitoring to more quickly detect and diagnose the cause of service failure.”
Challenge

- Identify unexpected behaviors before failures occur

- We need:
  - A representation of the *expected* behavior
  - A representation of the *unexpected* behavior
General Idea

1) Periodically collect performance metrics from a running system (black box analysis)

2) Build a statistical representation of the normal behavior of the system

3) Use a classifier to categorize behavior into normal or anomalous
Current Approaches

• Analyze system metrics independently [3]
  ✗ Anomalous combination of multiple metrics

• Find anomalies at a time instant [2, 6]
  ✗ Some anomalies take place over time (e.g., oscillation)

• Require historical failure data [5]
  ✗ Might be difficult to obtain
Our Approach

- Behavioral Analysis:
  - ✔ All metrics together
  - ✔ Over a time period

- Two-step Online Classification:
  - ✔ No need for historical failure data
  - ✔ Detects previously unseen anomalies
  - ✔ Identifies the type of the anomaly
Experimental Analysis

- Synthetic distributed mutual exclusion protocol
  - Several clients share access to various DBs
  - Based on a voting mechanism

- Three anomalous behaviors:
  - Deadlock, livelock, and oscillation

- 63 performance metrics:
  - Collected every second
  - CPU, memory, network, and disk usage per node
Evaluation Metrics

\[
\text{Recall} = \frac{\# \text{ of true positives}}{\# \text{ of positive observations}}
\]

Measures the anomaly detection accuracy

\[
\text{Precision} = \frac{\# \text{ of true positives}}{\# \text{ of positive predictions}}
\]

Measures the proportion of correct alerts

PERFECT SCENARIO = Recall = Precision = 1
Two Anomalies
Conclusions & Future Work

• Overview:
  – Black box analysis
  – Behavior over a time period
  – No historical failure data

• Overall detection ~85%
  – Depends on the impact in the metrics (need to quantify)

• Too many false positives
  – Due to the lack of historical failure data

• Try with a real scenario
References


Two-step Classification

Behavior Extractor

One-class Classifier
- normal or anomalous

Multi-class Classifier

Eval 1:
- N or D
  - not D
  - not N

Eval 2:
- N or L
  - not L
- D or L
  - not N
  - not D
  - not L

Eval 3:
- N or T
- L or T
- D or T

Result: normal (N)  livelock (L)  thrashing (T)  deadlock (D)
Impact on Network

![Graph showing network impact](image)

- Impact at start of starvation: 134 seconds
- Impact at end of starvation: 625 seconds

- Impact at start of deadlock: 134 seconds
- Impact at end of deadlock: 624 seconds