Automatic Detection of Runtime Performance Bugs in Cloud-like Environments

Zachary Parks, WeiRui Wang, Lalit Bangad
Motivation

• General: Runtime bugs
  – major challenge for large-scale, complex distributed systems

• Specific: Timeout and data-corruption bugs
  – sources of runtime performance degradation

• Crucial to understand the nature and causes of these bugs through practical approaches for detecting, diagnosing, and resolving them
Timeout Bugs

• Cause
  – System fails to respond within timeout period
  – Improper use of timeout parameters

• Impacts
  – Server systems hanging
  – Performance degradation
Detecting Timeout Bugs

Following the procedure similar to one outlined in TScope\textsuperscript{1}:

- When a performance bug occurs, we will use LTTng to retrieve a window of system call traces
- We will then extract the system calls related to timeout bugs
- Use total execution time (provided by LTTng trace windows) to generate feature vectors
- Use unsupervised behavior learning to detect execution time anomalies
  - Determine if, when an anomaly is found, the anomaly involves timeout-related system calls
Data Corruption Bugs

• Cause
  – Data stored or processed by a system becomes incorrect or inconsistent

• Impacts
  – Hanging and performance degradation
  – Complete outage/unresponsiveness
Detecting Data Corruption Bugs

Following the procedure outlined in DScope\(^2\):

- We will use the Soot compiler framework to generate IR from Java application bytecode
  - Then, we will traverse the control flow graph (CFG) to perform loop path extraction
  - Next, identify I/O dependent loops
  - Loop stride and bound analysis
Things to Learn and Do

- Building experimental environments (VCL):
  - Hadoop Common
  - HDFS
  - MapReduce
  - Soot compiler framework
- Replicating bugs and data collection:
  - BugZilla, Apache Jira
  - LTTng kernel tracing
- Unsupervised behavior learning:
  - SOM?
  - Restricted Boltzmann Machine?
References


Thank you!

Questions?