Pinpoint: Problem Determination in Large, Dynamic Internet Services

Mike Y. Chen, Emre Kiciman, Eugene Fratkin, Armando Fox, Eric Brewer
Univ of CA, Berkeley
Stanford Univ

Presented by
Ophelia Chesley
Motivation

• Requests for internet services very often travel through many components:
  – front-end load balancers
  – web servers
  – Frameworks
  – Databases
  – Everything else in between
• Hard to determine the cause of unanticipated faults among components
• Need to develop dynamic analysis methodology to detect problems and isolate root causes
Goals of the Methodology

• To dynamically trace internet requests through a multi-layered system without human intervention.
• To record the outcome (success or failure) and the components that service them.
• Use data clustering and statistical techniques to correlate failure to components.
Overview

• Assumptions
• Pinpoint Framework
  – Client request tracing
  – Failure Detection
  – Analysis
• Implementation
• Experiments
• Limitations of Pinpoint
• Conclusion
Assumptions

• Different combinations of components are used for different requests
  – Granularity at the component level

• Failure of a request is independent of the activities of other requests
  – Highly replicated internet services clusters
  – Minimized single-point of failure
Client Request Tracing

• Each request is assigned a requestID
• Instrument the middleware and communication layers to record <requestID, component_ID> pairs
• Request ID is passed from component to component by the middleware along with the call data
• Collect machine, cluster, component, component version, database table, configuration file information
• Depends on inter-component communication protocol
• Generates the trace log
Failure Detection

• Internal Failure Detection
  – Report failures that might be masked
  – Options to track assertions and exceptions generated by the application components

• External Failure Detection
  – Failures that are visible to users
  – Include infrastructure and application failures

• Generates the failure/success log
Data Analysis

• Use data clustering algorithm
  – Groups similar data points together
  – Correlates failure with a set of components cluster
Implementation of Pinpoint

- J2EE instrumentation
- Layer 7 Packet Sniffer
- Data Analyzer
J2EE Instrumentation

• Pinpoint sits on top of J2EE middleware
• No modifications at application level
• Instruments three types of components:
  – Enterprise JavaBeans (business/application logic)
  – Java Scripting Pages (dynamic HTML)
  – JSP tags that extends JSP
• Can be extended for any J2EE applications
J2EE Instrumentation

• Request ID stores in a thread-specific local variable
  – Assume components do not create threads
  – Does not support clustering
• Internal fault detector logs exceptions that pass component boundaries
• Request ID is passed to the external fault detector using the HTTP header
Layer 7 packet Sniffer

• Snifflet is the external fault detector
  – Built to capture TCP packets
  – Monitor TCP and HTTP failures (timeouts, resets, 404 not found, 500 Internet server error, etc.)
  – Can be programmed for customized failure detection
Data Clustering Analyzer

• Use hierarchical clustering method
  – Unweighted pair-group method using arithmetic averages (UPGMA)
    • Distance between clusters = average distance among all pairs of points within the clusters
  – Jaccard similarity coefficient
    • Distance between 2 points = number of requests they appear in together/all the requests the 2 points appear in total
Experiments

- System setup
- Metrics: accuracy versus precision
- Results
Experiments - Setup

• One machine with J2EE server
• One machine with a client browser emulator
• Use the PetStore demo application
• Executed 133 tests
  – Application server restarted after each test
  – 1 transaction active at any time
  – Each transaction exercise different sets of components
Experiments - Setup

• Each test is injected with faults:
  – Declared exceptions (masked by application)
  – Undeclared exceptions (often caught by middleware)
  – Infinite loops (TCP timeouts)
  – Null calls (detectable through other faults)

• Fault is always injected to the last component used in a request
Experiments - Metrics

• Accuracy
  – How often all components causing a fault are correctly identified

• Precision
  – Ratio between correctly identified faults and predicted faults

• Increasing accuracy can result in many false positives
Experiments - Results

• Compare Pinpoint to 2 other techniques (detection and dependency checking) in terms of accuracy and precision

• Detection:
  – Similar to Pinpoint’s internal fault detector
  – Return component where a failure is manifesting

• Dependency Checking:
  – Returns components that the failed requests used
  – Ignore successful requests
Experiments - Results

• Online overhead is about 8.4%
• Pinpoint has higher accuracy and precision than the other techniques
Experiments - Results

- Pinpoint does well for single-component failures
Experiments - Results

- Effectiveness of Pinpoint decreases as fault length increases.
- Fault length is the number of interacting components that causes a failure.
Experiments - Results

- Dependency always has low precision
- Not affected by fault length
Experiments - Results

- Detection is extremely sensitive to fault length
- Precision is about 30%
- Accuracy ranges from 50% to 0%
Limitations

• Pinpoint cannot distinguish tightly coupled components
• Pinpoint does not work for non-independent requests
• Pinpoint does not distinguish between bad inputs and failures
• Pinpoint does not capture masked faults that result in decrease in performance
Conclusion

• Presents a new problem determination framework for large, dynamic system
• Prototype Pinpoint has higher accuracy and precision that 2 other traditional techniques