Detecting and Diagnosing Application-Level Failures in Internet Services

Emre Kıcıman and Armando Fox
{emrek, fox}@cs.stanford.edu
Software Infrastructures Group
Stanford University
Outline

- Motivation for better detection & diagnosis
- Overview of Pinpoint analysis approach
- Details and results
- Integration efforts
- Related work & summary
Motivation

- TT Detect and Diagnose is a major component of MTTR
  - MTTR = TT Detect + TT Diagnose + TT Repair
  - One commercial svc estimates TT Detect is 75% of their MTTR
  - Study of 3 Internet services: better monitoring/detection would mitigate or avoid 65% of user observed failures [Oppenheimer]

- App-level failures
  - >60% of sites have user-visible (incl. app-level) failures [BIG-SF]
  - App-level failures can be expensive: lost customers, revenue, etc.

- Goal: App-generic & app-level failure detection
  - App-generic: low-maintenance, low cost.
  - App-level failures: problems that customer sees, but admin doesn't.
  - To repair: often need only location of fault
Ex. Application-Level Failure

No itinerary is actually available on this page

Ticket was bought in March for travel in April

But, website (superficially) appears to be working. Heartbeat, pings, HTTP-GET tests are not likely to detect the problem
Pinpoint Approach

■ Existing techniques
  ● Heartbeats/resource monitors: generic but miss high-level faults
  ● End-to-end tests: high-level, app-specific; $$ to develop/maintain.

■ Insight: many app-level faults will also cause anomalies in observable app structure & behavior
  1. Monitor low-level behaviors that imply some high-level semantics
  2. Compare behavior to believed-good behavior, watch for anomalies
  3. Correlate likely failures of requests with likely causes.

■ Some behaviors we monitor:
  ● Component behaviors: naming, resource, config problems
  ● Structure of runtime paths: cross-component bugs
  ● Data access patterns: errors in workflow across requests
Detection & Diagnosis w/Path Analysis

1. **Trace** execution path of every client request
   - Captures a vertical slice of system behavior
   - Modify middleware to record components, resources, latencies used per request
   - Aggregate many traces -> whole system behavior

2. Anomaly detection to detect failures
   - Extract an aspect of behavior that corresponds to app behavior
   - Get good-behavior from past behavior or replicated peers
   - Use statistical analysis to look for patterns and anomalies

3. Correlate failures to possible causes
   - For when unit of detection not same as unit of recovery
   - Use data clustering, decision tree learning.
Testbed and Faultload

- Instrumented JBoss/J2EE middleware
  - J2EE: state mgt, naming, etc. -> Good layer of indirection
  - JBoss: open-source; millions of downloads; real deployments
  - Track EJBs, JSPs, HTTP, RMI, JDBC
  - Performance: 2-40ms latency hit; 17% throughput decrease.

- Testbed applications
  - Petstore 1.3; Petstore 1.1 (clustered), ECPerf, RUBiS

- Test strategy: inject faults, measure detection rate
  - Declared exceptions: App should handle gracefully
  - Undeclared exceptions: App may not handle well
  - Omitted calls: extreme case of byzantine failure
Component Behavior Anomalies

- From traces, extract calls in/out of components
  - Weight each link according to proportion of calls passing through it.

- Anomalies in component behavior can indicate:
  - Internal failures: corrupt state, resource limits, buggy logic
  - External failures: naming service problems, illegal arguments
CB: First Results

- Overall, detected 64% of failures
  - Injected 3 kinds of failures into each of 47 components
  - Historical analysis for anomaly-detection
  - About 25-30% precision

- Analysis:
  - Changes in workload mix cause false positives
  - Hard to detect misbehaviors in simple, leaf EJBs
  - Hard to distinguish faults in tightly coupled components

---

**Expected** | **Unexpected** | **Omitted**
--- | --- | ---
50% | 70% | 30%
Improving CB-Analysis

- Workload mix changes cause false positives
  - Separate paths by request type before anomaly detection
  - Significant improvements to precision
  - URL is decent classifier of request type, but doesn't capture enough (e.g., whether user is logged in)

- Detecting misbehaviors in simple EJBs
  - Partial: more instrumentation of middleware services (JNDI)
  - But problem remains: simple EJBs have less behavior to analyze.

- Improving core algorithms
  - Studying machine-learning techniques for anomaly detection
Other Behaviors & Anomalies

- **Runtime path structures**
  - Analyze call-graph and resource usage of requests
  - Captures: buggy comp. interactions, request arguments, etc.
  - Correlate features of anomalous paths to locate “causes”
  - Results: 90% detection rate; 80% location rate
  - But only 50% end-to-end detection & location (?)

- **Patterns in persistent data accesses**
  - Analyze how DB records are accessed by requests
  - E.g., trace “PurchaseOrder” creations/reads/modifications
  - Captures: problems in high-level workflow
  - Status: under implementation
Performance Anomalies

- Component performance anomalies
  - Resource-leaks; errant background processes, etc.
  - Often early indicator of fail-stop problems

- Similar to component behavior analysis
  - But also look for deviant latency means and std. devs.
  - Harder to compensate for workload variations

- Status
  - Testing with various performance failures
  - Analyzing initial results
Integration Efforts

- JBoss with Application-Generic Recovery (JAGR)
  - On-line Pinpoint monitor is integrated with micro-reboot & AFPI
  - Lower cost of false positives -> enables more aggressive PP
  - Running experiments to compare performability improvements

- Session State Management (SSM)
  - Detect performance and behavior anomalies in SSM bricks
  - Capture statistics on msg recv/send/drop, queue length, etc.
  - Initial experiments promising.

- Summer '03 (MSR): Configuration errors in app clusters
  - At MSR: work with Yi-Min Wang to combine Strider analysis of windows registry with PP-style detection & diagnosis
Related Work

- Detection and diagnosis:
  - Richardson: Performance failure detection
  - Infospect: search for logical inconsistencies in observed configuration
  - Event/alarm correlation systems

- Compilers & PL
  - DIDUCE -> hypothesize invariants, report when they're broken
  - Liblit & Aiken: across installed base, correlate crashes w/state
  - Engler: analyze static code for patterns and anomalies -> bugs

- Request Tracing
  - Magpie: tracing for performance modeling/characterization
  - Commercial products for debugging distributed J2EE apps
Summary

- **App-generic failure detection and diagnosis**
  - Monitor low-level behaviors that imply high-level semantics
  - Watch for anomalies in captured behaviors
  - Correlate likely failures to possible causes

- **Current status:**
  - Analyzed initial results; now iterating and improving PP.
  - Testing JAGR, SSM integration
  - Using machine learning algorithms for improved det & diag

- **Immediate future:**
  - Diagnosing configuration errors
More Information

http://pinpoint.stanford.edu/