Pinpoint

Status and Future Directions

Emre Kılınçar
emrek@cs.stanford.edu
http://pinpoint.stanford.edu/

Winter ROC Retreat,
January 14, 2003
Pinpoint Overview

- **Macro Analysis**
  - Runtime paths: Capture component interactions
  - Statistical analysis: Capture aggregate behavior

- **System Requirements**
  - Fine-grained units of work (e.g., a request)
  - Traceable path through system for each request
  - Large numbers of requests

- **Our Goal: Reduce time to detect and diagnose failures**
  - MTTR = Time to (detect + diagnose + repair)
  - Focus on J2EE / JBoss applications
Outline

- Pinpoint v2 Architecture
- Detecting Anomalies
- Status
- Integration
- Summary
Review: Pinpoint v1

- Fault diagnosis via data clustering
  - Off-line analysis; no *a priori application model*
- Sun's J2EE reference implementation
  - Single-node; log EJB, JSP, and JSP tags
- Results: trade-off accuracy against false-positives
  - Accuracy: 70-90%. False-positives: 20-40%
  - Other techniques: Either have poor accuracy (40%) or many false-positives (90%)
On to Pinpoint v2

- For Pinpoint v2:
  - On-line analysis
  - Instrument more robust, clustered J2EE platform (JBoss)
- Tools to attack wider range of problems
  - Deducing application structure
  - Detecting application-level faults
- Improve fault diagnosis for multiple component faults
- Integration with repair processes
Pinpoint v2 Architecture

- JBoss System
- JBoss Tracer
- JBoss Collector

Analysis Engine
- Store
- Sorting
- Clustering
- ... 

Input Plugins

Data Analysis Plugins

Output Plugins
Architecture: Tracing JBoss

- Instrument JBoss middleware
  - Modify HTTP server, wrappers for EJB, JSP, JDBC
  - Observe calls to and returns from components, exceptions
  - Record component details, SQL queries, timestamp
  - Record path context: request id, sequence number
- Challenge: keeping track of requests
  - Intra-thread: Keep request id in thread-specific location
  - Cross-threads: Modify RMI, HTTP to forward request id
- Total: added 428 lines in 10 files
Architecture: Plugins

- Sorting observations
  - By request id -> runtime path
  - By component identifiers -> all info about component
    - Note: get to choose what defines a component.
  - By link id (src/sink component identifiers).

- Statistics plugin
  - Calculates simple statistics about sets of observations

- Plus others
  - Storage, HTTP frontend, data clustering
Anomaly Detection

- What is an anomaly?
  - Paths: deviant *structures* or latencies
  - **Components**: performance/behavior variation
  - Compared to historical norms, or current peers

- Generic method of detecting likely errors

![Diagram](database-component-variation)
1. Generate path traces from observations
2. Separate paths by request type
Detecting Anomalies in Paths (3)

3. Cluster similar structures together

Type A

Type B
Detecting Anomalies in Paths (4)

4a. Peer comparison: differences between clusters can be considered anomalies
   - But, differences are often normal

4b. Historical comparison: compare number, structure and size of clusters to history
   - More robust, but need to be able to identify requests over time
Detecting Anomalies in Component

- Component-based failure detection
  - Group observations by component
  - Find peers, based on identifying attributes
  - Calculate links starting/ending at peer components
  - Compare links (latency & number of requests)
  - Significant variation from norm is an anomaly
Anomaly Detection Status

- Component-based failure detection
  - Built, currently testing and debugging
- Request-based failure detection
  - In development stage
  - Challenge: when to use historical vs. peer comparison
Status

- Deriving application structure
  - State dependency & component call graphs working
- Correlating failures
  - Reimplemented Pinpoint v1 algorithm
  - Improving performance for on-line usage
- Testbed
  - Clustered Petstore v1.1.2
  - ECPerf benchmark
  - Looking for more apps & real environments
Plans: Integrating with Repair

- ... with human repair
  - Visualization of system structure, likely failures and causes
  - Paths have direct visual representation

- ... with automatic repair
  - Planning integration with Recursive Recovery
  - Send high-confidence diagnoses to RR for restart
Summary

- **Goals of Pinpoint v2**
  - On-line analysis of clustered J2EE applications
  - Deduce application structure, detect failures, diagnose causes, without *a priori application models*

- **Work-in-progress**
  - Built tracing, extensible framework for analysis
  - Developing anomaly detection
  - Improving fault correlation performance
  - Initial results from macro-analysis are promising
http://pinpoint.stanford.edu/


*Using Runtime Paths for Macro-Analysis.* Mike Chen, Emre Kılıçman, Anthony Accardi, Armando Fox, Eric Brewer. In submission.

Plus presentations...
Extra Slides
Focus on J2EE/JBoss

- J2EE
  - [what is j2ee]

- Jboss
  - [what is jboss]

- Pinpoint
  - Tracing Apps in Jboss
  - Analyzing separately, platform-generic analysis engine
Architecture: Analysis Engine
Clustering Requests

- Calculate distance between two sets of observations
  - Identify aspect of observation that we care about (request id, component id, etc.) (note, boolean valued)
  - For each unique aspect, see whether it is true in both requests, false in both, or true in one and false in the other.
  - Distance coeff: jaccard coeff.

- Cluster hierarchically, merge two closest clusters until everything is a single cluster, or distance reaches threshold
Deducing Application Structure

- Application Structure definition
  - [Generally, why it's important]
  - [including: “we use it for later analyses”]
- State dependencies
  - Associate external request w/internal data
- Component call graphs...
  - Which components call each other
- Request, component classification (magpie)
  - What requests are similar
Deducing App Structure [How]

- REMOVE ?
- 1. Organize observations by request id -> path
  - Cluster paths to find similar requests
- 2. Organize observations by component identifiers
  - <component name, ipaddr> for physical instances,
  - <component name> for logical instances
  - Cluster components to find ones used together.
- ... state dependency
Deducing App Structure

- REMOVE ? (add graph and move to extra slides)
- Initial results
  - [state dependency graph for pinpoint]
  - [graph of part of a request's call graph]
- Re-assert that we'll use this information later when tracking bugs.
Diagnosis

- REMOVE

- What is “diagnosis”
  - Try to find cause of fault (however we detected the fault)
  - By correlation: what observations correlate highly with faults? e.g., most requests that use data X fail.
  - Highly correlated observations are likely to be either side-effects of a fault or causes of a fault.

- Why do we care?
  - If we have a good guess at the failure, we can try to
Diagnosis

- REMOVE
- [How – maybe leave this out, since we talked about PP last year]
- For all requests, both successful and failed, cluster
Diagnosis

- REMOVE

- Initial Results (from Pinpoint paper)
  - Summarize, [pull numbers from photos paper]

- Plans:
  - Porting to work on-line.
  - Improve detection for multi-component faults
Architecture: Observations

- Observation: Discrete unit of tracing
  - Before and after calling component, accessing data
  - Error occurs

- Record...
  - Details: component name, version, SQL query, ...
  - Path info: request id, sequence number
  - Timestamp