Pythia: A Just-in-Time Instrumentation Framework for Distributed Systems

Lily Sturmann☆△, Emre Ates+, Peter Portante△, Orran Krieger☆+△, Ayse Coskun+, Raja Sambasivan☆+
Massachusetts Open Cloud☆ / Boston University+ / Red Hat△
You → Time → Distributed System

Emotionally evocative weather
Unique challenges in debugging distributed systems

Where is the problem?
It could be in ...

- One of many components
- One of several stack levels
- Inter-component interactions
Adventure 1: Painful Debugging Cycle

The instrumentation in your system does not give enough information about the root cause of the problem. You can add instrumentation. 

Where do you add instrumentation?

Problem found? Usually no.....

Gather data from current instrumentation

Use data to guess where to add instrumentation
Adventure 2: Using Pythia to Automate Debugging Cycle

- Before the engineer, Pythia investigates the traces
  - Locates problematic areas
- Enables instrumentation that would help locate problems
  - Disables “boring” instrumentation
- Repeats the cycle
  - Enables deeper instrumentation that is closer to the problem
- Engineer only inspects lean traces that include information about the potential problem
  - Only need to fix the problem
  - Skipped the painful cycle
Pythia’s key insight: Performance variations

- **High perf. variation** among workflows expected to perform similarly
  - Indicative of problems!
- **Localizes perf. variation** to identify where additional instrumentation needed
- Enables **necessary instrumentation** points in system
- **Repeats** as necessary
Pythia’s key enabling technology: end-to-end tracing

- Captures request workflows by sending metadata with workflows
- Tags enabled instrumentation points with workflow metadata
- Re-constructs workflow as trace by stitching together instrumentation data relating to one request
Pythia Workflow

1. Gather current request skeletons
2. Extract critical path from skeletons
3. Group critical path skeletons based on similarity
4. ID groups with high perf. variation
5. Localize perf. var
6. Determine what instrumentation to enable
4. ID consistently slow groups

Perf. NOT similar
Pythia’s current status

- Exploring applicability to OpenStack and eventually Ceph
- Augmenting OpenStack’s tracing system to support Pythia
- Prototype codebase identifies & localizes high perf. Variation
- Initial testing on simple OpenStack problems

Some research questions

- Are there other metrics that can predict where instrumentation may be needed?
- What is the minimum level of instrumentation for Pythia to begin its analysis?
  - What are the stopping conditions for the analysis cycle?
- What is the relationship between the search strategies used for enabling instrumentation and Pythia’s output?