An Automated, Cross-Layer Instrumentation Framework for Diagnosing Performance Problems in Distributed Applications

By Emre Ates¹, Lily Sturmann², Mert Toslali¹, Orran Krieger¹, Richard Megginson², Ayse K. Coskun¹, and Raja Sambasivan³

¹Boston University; ²RedHat, Inc.; ³Tufts University

ACM Symposium on Cloud Computing
November 21, 2019, Santa Cruz, CA
Debugging Distributed Systems

Challenging: Where is the problem?
It could be in:

- One of many components
- One of several stack levels
  - VM vs. hypervisor
  - Application vs. kernel
- Inter-component interactions
Today’s Debugging Methods

Different problems benefit from different instrumentation points.

You can’t instrument everything: too much overhead, too much data.
Gather data from current instrumentation

Able to identify problem source? Usually no…

Use data to guess where to add instrumentation

Today's Debugging Cycle
Our Research Question

Can we create a continuously-running instrumentation framework for production distributed systems that will automatically explore instrumentation choices across stack-layers for a newly-observed performance problem?
Key insight: Performance variation indicates where to instrument

- If requests that are expected to perform similarly do not:
  - There is something unknown about their workflows, which could represent performance problems
  - Localizing source of variation gives insight into where instrumentation is needed.

A READ request from storage
Request #2
Request #3
**Key Enabler: Workflow-centric Tracing**

- Used to get workflows from running systems
- Works by propagating common context with requests (e.g., request ID)
  - Trace points record important events with context
- Granularity is determined by instrumentation in the system
Vision of Pythia

Gather data

Gather workflow skeletons
Vision of Pythia

Gather data

Gather workflow skeletons

Extract critical paths
Vision of Pythia

Gather data

Gather workflow skeletons

Extract critical paths

Group workflows by performance expectation
Vision of Pythia

Gather data
- Gather workflow skeletons
- Extract critical paths
- Group workflows by performance expectation

Identify problematic groups
- Identify groups with high performance variation
Vision of Pythia

Gather data
- Gather workflow skeletons
  - Extract critical paths
    - Group workflows by performance expectation
  - Identify groups with high performance variation
- Identify problematic groups
- Determine where to enable instrumentation
  - Localize performance variation
Vision of Pythia

Gather data
- Gather workflow skeletons
  - Extract critical paths
    - Group workflows by performance expectation

Identify problematic groups
- Identify groups with high performance variation

Determine where to enable instrumentation
- Localize performance variation

Determine what instrumentation to enable
- Enable/disable additional instrumentation
Vision of Pythia

- Gather data
  - Gather workflow skeletons
  - Extract critical paths
  - Group workflows by performance expectation

- Identify problematic groups
  - Identify consistently slow groups
  - Identify groups with high performance variation

- Determine where to enable instrumentation
  - Localize dominant contributors to response time
  - Localize performance variation

- Determine what instrumentation to enable
  - Enable/disable additional instrumentation
Vision of Pythia

Gather data
- Gather workflow skeletons
- Extract critical paths

Identify problematic groups
- Identify consistently slow groups
- Identify groups with high performance variation

Determine **where** to enable instrumentation
- Localize dominant contributors to response time
- Localize performance variation

Determine **what** instrumentation to enable
- Enable/disable additional instrumentation
Vision of Pythia

Gather data
- Gather workflow skeletons
- Extract critical paths
- Group workflows by performance expectation

Identify problematic groups
- Identify consistently slow groups
- Identify groups with high performance variation

Determine where to enable instrumentation
- Localize dominant contributors to response time
- Localize performance variation

Determine what instrumentation to enable
- Enable/disable additional instrumentation
Vision of Pythia

Gather data
Gather workflow skeletons
Extract critical paths
Group workflows by performance expectation

Identify problematic groups
Identify consistently slow groups
Identify groups with high performance variation

Determine where to enable instrumentation
Localize dominant contributors to response time
Localize performance variation

Determine what instrumentation to enable
Enable/disable additional instrumentation
Challenge 1: Grouping

Gather data
- Gather workflow skeletons
- Extract critical paths
- Group workflows by performance expectation

Identify problematic groups
- Identify consistently slow groups
- Identify groups with high performance variation

Determine where to enable instrumentation
- Localize dominant contributors to response time
- Localize performance variation

Determine what instrumentation to enable
- Enable/disable additional instrumentation
Which Requests are Expected to Perform Similarly

- Depends on the distributed application begin debugged
- Generally applicable: Requests of the same type that access the same services
- Additional app-specific details could be incorporated

Expectation 1: Read requests
Expectation 2: Auth requests
Challenge 2: Localization

- Gather data
  - Gather workflow skeletons
  - Extract critical paths
  - Group workflows by performance expectation

- Identify problematic groups
  - Identify consistently slow groups
  - Identify groups with high performance variation

- Determine where to enable instrumentation
  - Localize dominant contributors to response time
  - Localize performance variation

- Determine what instrumentation to enable
  - Enable/disable additional instrumentation
Localizing Performance Variations

- Order groups and edges within groups.
  - How to quantify performance variation?
- Multiple metrics to measure variation
  - Variance/standard deviation
  - Coefficient of variance (std. / mean)
    - Intuitive
    - Very small mean -> very high CoV
  - Multimodality
    - Multiple modes of operation
Challenge 3: What to enable

Gather data
- Gather workflow skeletons
- Extract critical paths
- Group workflows by performance expectation

Identify problematic groups
- Identify consistently slow groups
- Identify groups with high performance variation

Determine where to enable instrumentation
- Localize dominant contributors to response time
- Localize performance variation

Determine what instrumentation to enable
- Enable/disable additional instrumentation
Search Space

- How to represent all of the instrumentation that Pythia can control?
- How to find relevant next-trace-points after problem is narrowed down?
- Trade-offs:
  - Quick to access
  - Compact
  - Limit spurious instrumentation choices

Search Strategies

- How to explore the search space?
  - Quickly converge on problems
  - Keep instrumentation overhead low
  - Reduce time-to-solution
- Many possible options
  - Pluggable design
Search Space: Calling Context Trees

- One node for each calling context i.e., stack trace
- Leverages the hierarchy of distributed system architecture
- Construction: offline profiling
- Trade-offs
  - Quick to access
  - Compact
  - Limit spurious instrumentation choices
Search Strategy: Hierarchical Search

- One of many choices
- Search trace point choices top-down
- Very compatible with Calling Context Trees
Search Strategy: Hierarchical Search

- One of many choices
- Search trace point choices top-down
- Very compatible with Calling Context Trees
Search Strategy: Hierarchical Search

- One of many choices
- Search trace point choices top-down
- Very compatible with Calling Context Trees

![Diagram of search strategy]

Offline-collected trace:
- nova start
  - keystone start
    - keystone end
    - neutron start
      - neutron end
      - glance start
    - glance start
  - keystone
- glance start

Search space:
- nova
  - neutron
  - glance
  - keystone
  - neutron
Search Strategy: Hierarchical Search

- One of many choices
- Search trace point choices top-down
- Very compatible with Calling Context Trees
Search Strategy: Hierarchical Search

- One of many choices
- Search trace point choices top-down
- Very compatible with Calling Context Trees

Diagram:

- Offline-collected trace:
  - nova start
    - keystone start
    - neutron start
    - neutron end
  - glance start
  - keystone
  - glance

- Search space:
  - nova
    - neutron
    - glance
    - keystone
Explaining Variation Using Key-Value Pairs in Trace Points

- Canonical Correlation Analysis (CCA)
- Used to find important key-value pairs in the traces

\[ a' = \max_a \text{corr}(a^T X, Y) \]

\[ Y = (t_1, t_2, \ldots, t_n) \text{ the request durations} \]
\[ X = (x_1, x_2, \ldots, x_m) \text{ the collected variables} \]
\[ a' \in \mathbb{R}^m \text{ the coefficients indicating most correlated variables} \]
Vision of Pythia – Completing the Cycle

Gather data
- Gather workflow skeletons
- Extract critical paths
- Group workflows by performance expectation

Identify problematic groups
- Identify consistently slow groups
- Identify groups with high performance variation

Determine where to enable instrumentation
- Localize dominant contributors to response time
- Localize performance variation

Determine what instrumentation to enable
- Enable/disable additional instrumentation
Validating Pythia’s Approach

- Can performance variation guide instrumentation choices?
- Run exploratory analysis for OpenStack
  - Start with default instrumentation
  - Localize performance variation
  - Find next instrumentation to enable
  - Use CCA for finding important key-value pairs
Validating Pythia’s Approach - Setup

- **OpenStack**: an open source cloud platform, written in Python
- **OSProfiler**: OpenStack’s tracing framework
  - We implemented controllable trace points
  - Store more variables such as queue lengths
- Running on MOC
  - 8 vCPUs, 32 GB memory
- Workload
  - 9 request types, VM/floating IP/volume create/list/delete
  - Simultaneously execute 20 workloads
Step 1: Grouping & Localization

- Collect latency values for each request
- Grouping: Same request type with same trace points
- Server create requests have unusually high variance and latency
- Pythia would focus on this group
Step 2: Enable additional instrumentation

- Pythia localizes variation into a semaphore in server create
- After adding queue length variable into traces, we see 3 distinct latency groups
  - CCA also finds this variable important

**TAKEAWAY:** Pythia’s approach identifies the instrumentation needed to debug this problem
Open Questions

- What is the ideal structure of the search space? What are possible search strategies? What are the trade-offs?
- How can we formulate and choose an “instrumentation budget”?
- How granular should the performance expectations be?
- How can we integrate multiple stack layers into Pythia?
More in the paper

- Pythia architecture
- Problem scenarios
- Instrumentation plane requirements
- Cross-layer instrumentation
Concluding Remarks

- It is very difficult to debug distributed systems
- Automating instrumentation choice is a promising solution to overcome this difficulty

More info in our paper (bu.edu/peaclab/publications)
Please send feedback to ates@bu.edu or join us at the poster session