CATCH ME IF YOU CAN: PERFORMANCE BUG DETECTION IN THE WILD

Milan Jovic, Andrea Adamoli and Matthias Hauswirth

Performance testing

Determines how a system performs in terms of responsiveness and stability under a particular workload. Investigate, measure, validate or verify other quality attributes of the system, such as scalability, reliability and resource usage

- It can show that the system meets performance criteria
- It can compare two systems to find which performs better
- It can measure what parts of the system or workload causes the system to perform badly

Performance testing

- **Profilers** measure what parts of a device or software contributes most to the poor performance or to establish throughput levels (and thresholds) for maintained acceptable response time

Profilers

- Help developers to find and fix performance problems

- Used in dynamic program analysis for measuring:
  - Usage of memory
  - Usage of particular instructions
  - Frequency and duration of function calls, etc.

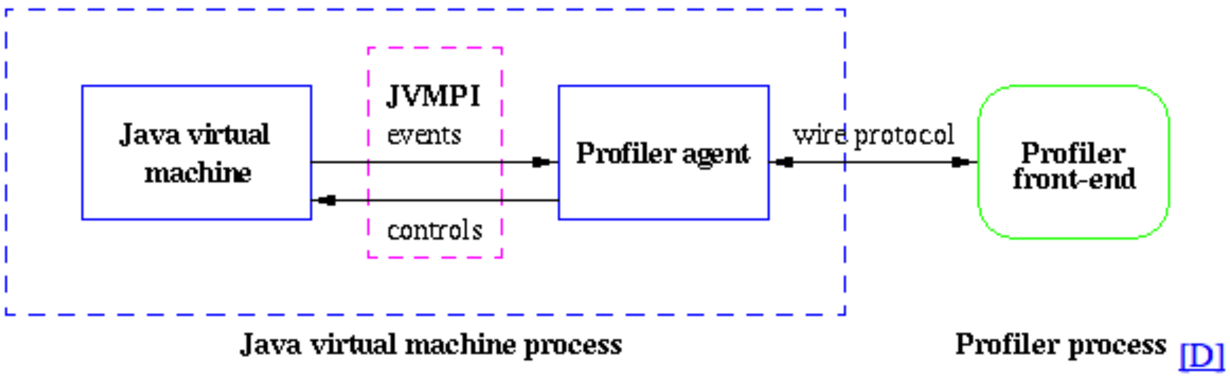
- Can be divided into:
  - Event based
  - Statistical
  - Instrumentation
  - Simulation

Event-based profilers

- .Net
- Ruby
- Phyton
- Java (Java Virtual Machine Profiler Interface (JVPI))

JVMPI

- Two-way function call interface between JVM and profiler agent
  - The virtual machine notifies the profiler agent of various events
  - The profiler agent issues controls and requests for more information
- A profiling tool based on JVMPI can obtain
  - Heavy memory allocation
  - CPU usage hot-spots
  - Unnecessary object retention
  - Monitor contention



Statistical profilers

- Probes the target's program counter at regular intervals using operating system interrupts
- Not exact, but it is a statistical approximation
- Allows the target program to run at near full speed
- Examples:
  - AMD CodeAnalyst
  - Apple Inc. Shark
  - gprof
  - Intel Vtune
  - Parallel Amplifier (part of Intel Parallel Studio)

Instrumenting profilers

- It instruments the target program with additional instructions to collect the required information

- Instrumenting will always have some impact on the program execution, typically always slowing it

- gprof is an example of a profiler that uses both instrumentation and sampling
  - Gathers caller information and the actual timing values are obtained by statistical sampling.

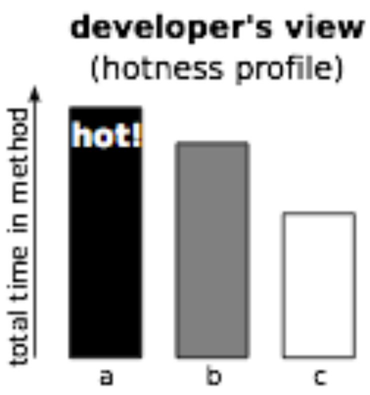
Performance Bug Detection in the Wild
Catching bugs in the wild

- Performance problems depends on the context in which an application runs
  - The underlying platform
  - The specific configuration of the application
  - The size and structure of the inputs the application processes
- Without knowing the exact usage scenarios of widely deployed applications, developers cannot conduct representative performance tests

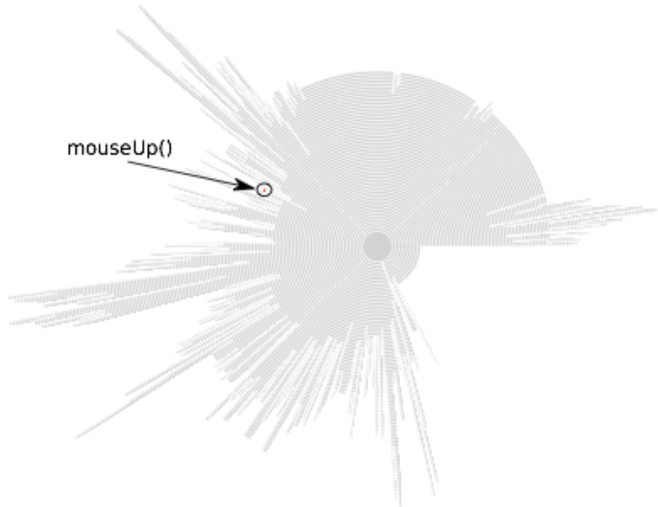
Performance Bug Detection in the Wild
Introduction

- Do the profilers find performance bugs - performance problems that real users actually notice?

- Profilers help developers to identify and optimize hot code



Performance Bug Detection in the Wild
Introduction



- Output of a traditional code hotness profiler in the form of a calling context tree
- The center of the diagram represents the application's main method
- The angle of a ring segment is proportional to the hotness of the corresponding calling context

Aim of the paper

- Automatically catch perceptible performance bugs by focusing on the latency instead of the hotness of methods

- Considerations:
  - Instrumentation and data collection can significantly slow down the application
  - Large overhead perturbs the time measurements and casts doubt on the validity of the measurement results

- A central goal of the paper's approach is thus the ability to **measure latency** and to gather actionable information about the reason for long latency method calls, while keeping the **overhead minimal**

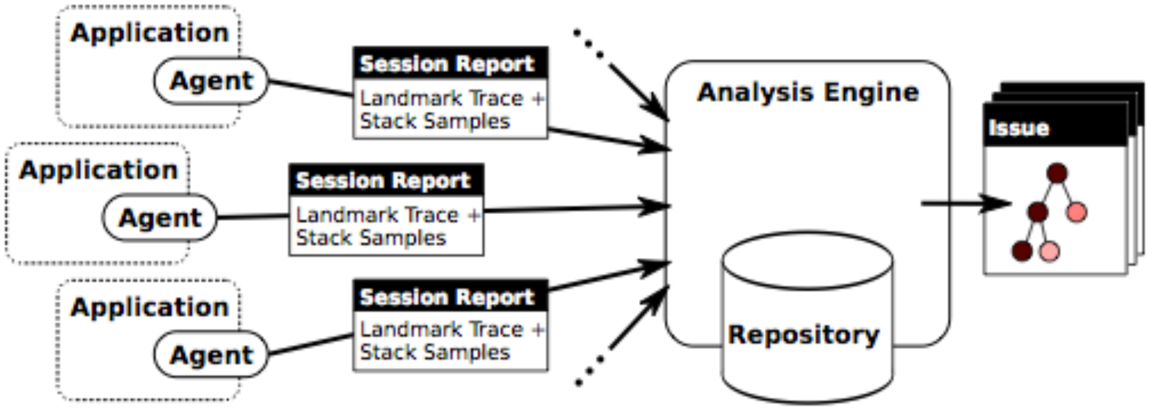
Approach

- Reduce overhead while still collecting the information necessary to find and fix bugs

- The collected information shall enable a tool to effectively aggregate a large number of session reports into a short list of relevant performance issues.
    - Each containing possibly many occurrences of long-latency behavior

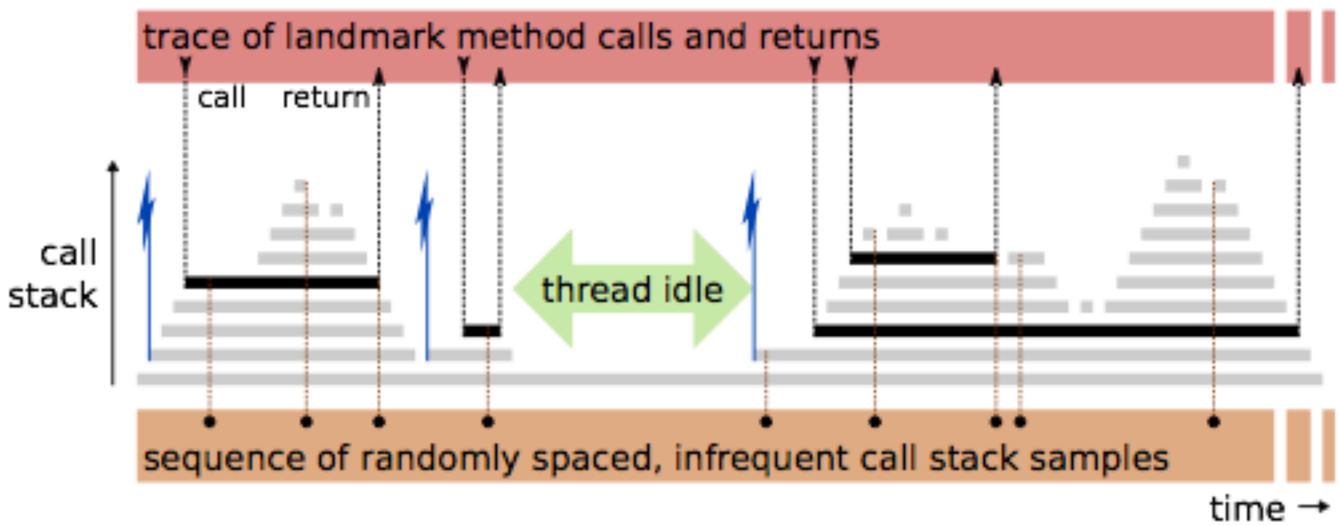
Overall view

- Applications are deployed with an agent that collects performance information about each session
- At the end of a session, the agent sends a session report to an analysis engine running on a central server.
- The server collects, combines, and analyzes the session reports to determine a list of performance issues



What to collect?

- It captures calls and returns from so-called landmark methods. It contains the timing information necessary to measure lag
- It captures rare, randomly spaced call stack samples of the running threads



Landmark methods

- Need to cover most of the execution of the program
- Need to be called during the handling of an individual user event
- Should be called infrequently
  - Tracing landmarks that are called large numbers of times for each user event would significantly increase the overhead

Landmark selection strategies

- Event dispatch method
  - This method will cover the entire event handling latency
  - If it is the only landmark, the analysis will result in a list with this method as a single issue

- Event-type specific methods
  - Low-level user actions (mouse move, mouse click, key press)
  - A mouse click will be handled differently in many different situations

Landmark selection strategies

- Commands
  - Different actions a user can perform in an application
- Observers
  - An application notifies any registered observers of its changes
- Component boundaries
  - For example, any call between different plug-ins could be treated as a landmark
- Application-specific landmarks
  - Select specific methods

Analysis

- The analysis engine extracts all landmark invocations from the landmark trace
  - **Inclusive latency** = landmark end time - landmark start time
  - **Exclusive latency** = inclusive latency - time spent in nested landmarks
  - The repository contains information about the distribution of its latencies, the number of occurrences, and the sessions in which it occurred
  - It updates the calling context tree related to that landmark
    - The tree is weighted with the number of samples in which it occurred.

Connecting issues & Call pruning

- It computes the similarities between the issues' calling context trees
  - Two different issues may trigger the same cause of long latency
  - It can simplify the search for the
  - It can help to prioritize the issue

- To test the impact of a fixed bug, developers would usually change the program and rerun it to confirm that the change indeed reduced the perceptible latency

Experimental setup

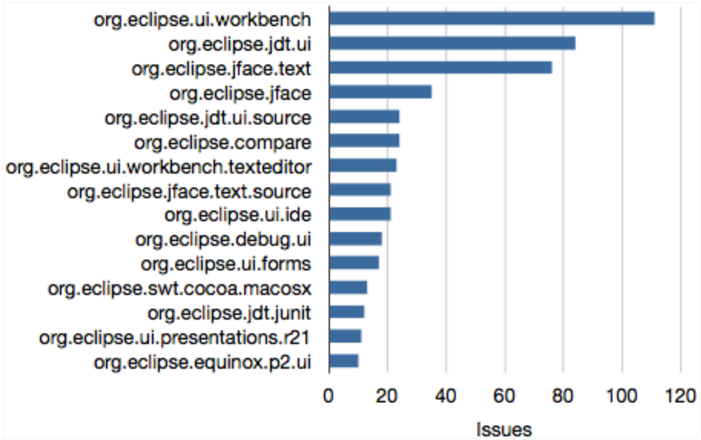
- Eclipse IDE
- 3 months - 1958 hours
- 1108 session reports
- 24 users
- Call stack samples taken every 500 ms
  - All samples taken during idle time intervals were dropped

Experimental setup Issue characterization

- Data per detected issue
  - Mean, first quartile, median, second quartile, 90-th percentile, and maximum value
  - Average exclusive latency: the longer its latency, the more aggravating the issue
  - Next three parameters describe how prevalent a given issue is
  - The last parameter, the number of collected call stacks, is indicative of the chance of fixing the issue

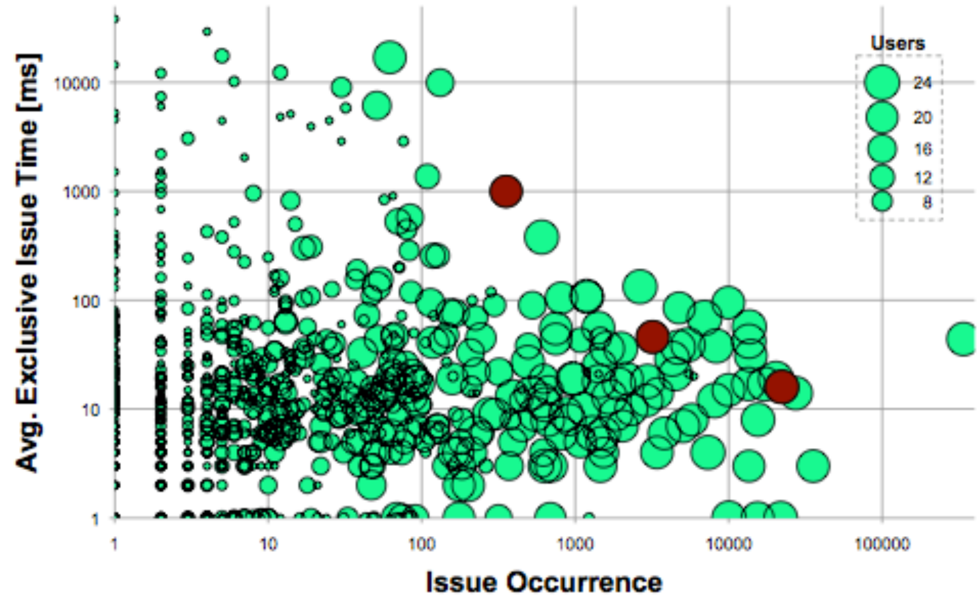
	Avg	Q1	Med	Q3	p90	Max
Avg.Excl.[ms]	320	4	11	31	119	38251
Users	5	1	2	8	19	24
Sessions	49	1	3	14	115	1069
Occurrence	896	2	6	48	449	338622
Stacks	65	0	0	3	42	28613

Results



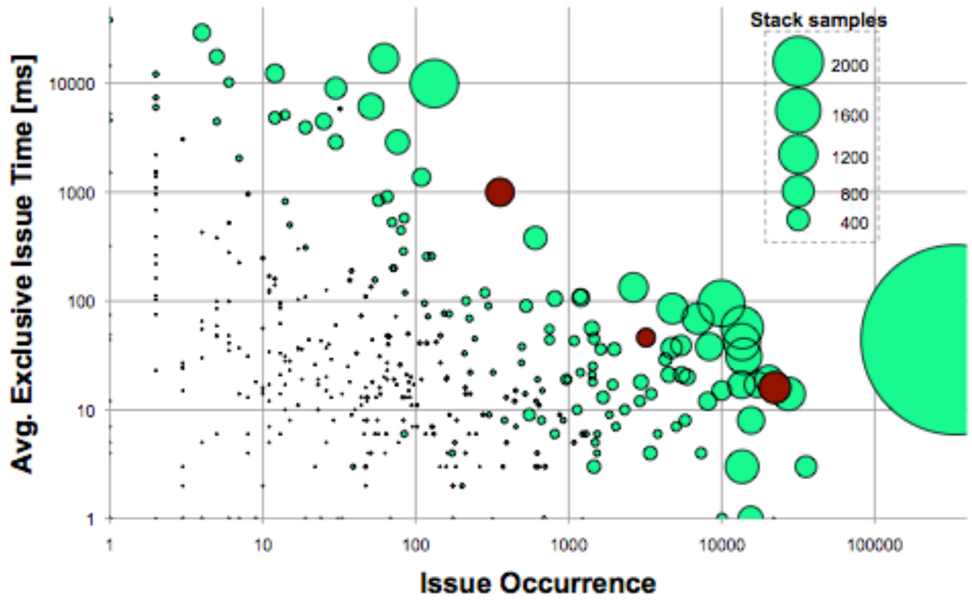
- Most of the reported landmarks correspond to listener notifications
  - 275 out of 881 are located outside the standard Eclipse classes
  - The majority of the 606 landmarks are located in a few dominating plug-ins
    - Eclipse workbench user-interface (111 landmarks)
    - Java Development Tools (JDT) user interface (84)
    - JFace text editor (76) plug-ins

Number of users encountering an issue



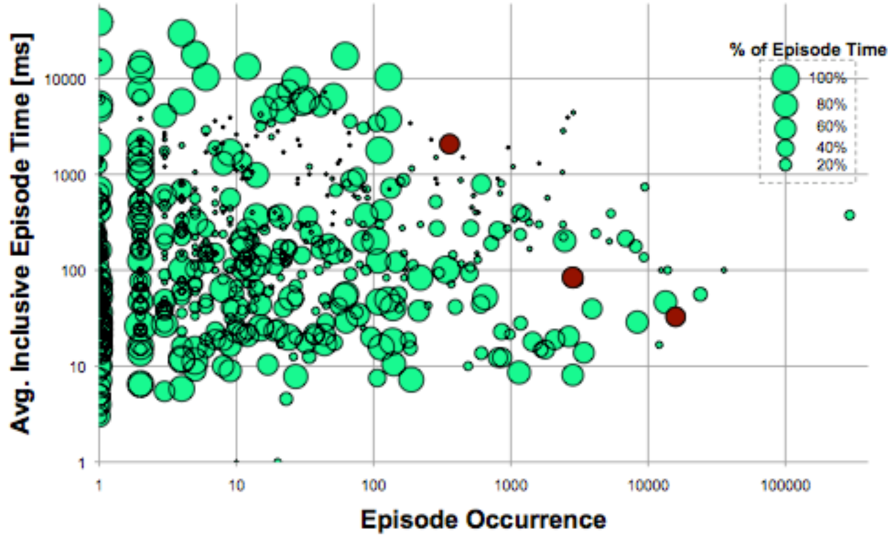
- Some long latency issues correspond to rare user operations
  - Project creation or
  - Activity in non-standard plug-ins (e.g. Android)

Number of collected stack samples



- Issues that occur often or exhibit a long latency have a higher chance of being encountered by the JVMTI stack sampler

Severity of issues



- It represents the the severity perceived by the user for a given issue
- x-axis represents how often a user is annoyed
- y-axis represents how much a user is annoyed each time
- Size of node: “I know this circle represents a frequent (x) and big (y) annoyance, but how much can I reduce this annoyance if I fix this issue?”

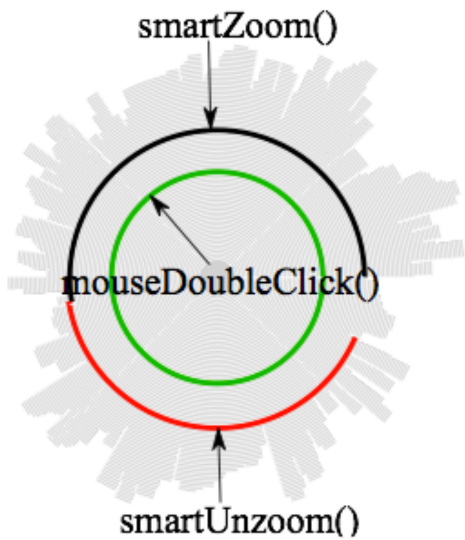
Issues under study

We consider three issues:

Landmark	Avg. Excl. Latency [ms] Identified	Reproduced	Fixed
mouseDoubleClick	999	149	4
verifyText	16	458	13
keyPressed	46	30	<3

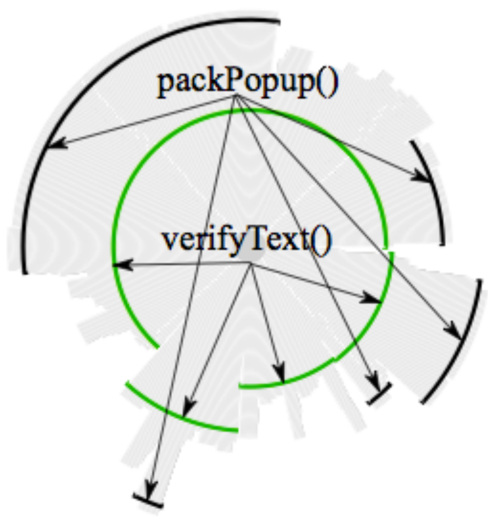
Maximize and restore

- The root of the tree represents the main method
- Each calling context is represented by a ring segment corresponding to the number of times a given calling context was sampled



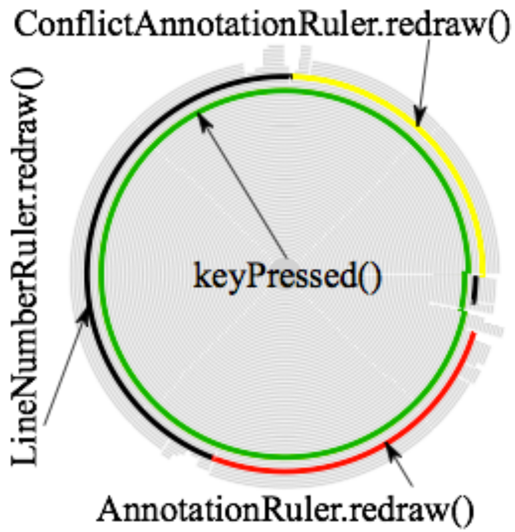
Rename refactoring

- The black ring segments represent the root cause of the problem
  - They correspond to calls to the packPopup method of the *RenameInformationPopup* class



Scrolling

- Most of the time in this issue is spent redrawing three components:
  - LineNumberRuler
  - AnnotationRuler
  - ConflictAnnotationRuler



Linking issues by similarity

"if you are interested in issue X, you may also want to have a look at issue Y"

- Comparison of the their calling context trees.

Limitations

- User-relevant issues
- False positives
- False negatives
- Concurrency
- No automatic optimization
- Difficulty of issue reproduction
- Cost of stack sampling
- Inaccuracy of stack sampling
- Limitations of call pruning

Conclusions

- In this work, the authors proposes a technique (and a tool named LagHunter) which focuses on latency bug detection in interactive applications

- Combines a low-overhead approach to latency profiling with call stack sampling

- Automatically computes information about similar latency bugs

- Performs semi-automated call pruning, to efficiently help the developers finding the causes of long latency