TSINGHUA UNIVERSITY

Summary on *Understanding and Detecting Real-World Performance Bugs*

XINGGAO YANG

LAST EDIT:February 26, 2017

1 Basic Info

Title: *Understanding and Detecting Real-world Performance Bugs*;
Author: G. Jin, L. Song, X. Shi, J. Scherpelz, S. Lu;
Other: PLDI'12, 2012;

2 My conclusion

This paper does give a **not complete and strict** defination of performance bugs, and awareness and concepts of their harm and features. They come up a way to detect them, but far from enough.

3 Abstract

This paper is about low efficient codes, and authors investigated 109 real-world performance bugs trying to summarize their pattern and find a way to detect them.
The so called *performance bugs* are hard to be found, but are able to lead to reduced throughput, increased latency and wasted resources in the field.
They showed that the bugs are caused by very short codes,

4 Introduction & Motivation

Performance bugs degrade performance significantly but are poorly understood, and simple source code fixing can significantly speed up software, while preserving functionality. Because performance testing relies on ineffective **black-box random testing** or **manual input design**, they are hard to be detected and most of them flee away. And as their non fail-stop symtoms, they are costly to be detected, and several months of time may be needed to find these bugs.
And the following trend will make the performance-bug problem more **critical** in the future:

- **Hardware**: Performance bugs will be more harmful when each core will not perform better in multi-core applications;
- **Software**: Software system will be more **complex** and handle more **workloads**, performance will suffer from severer degradation;
- **Energy efficiency**: Of course, performance bugs will lead to energy inefficiency. But it is interesting, I've never think about program energy problem before, and as the systems grow larger inefficiency will not be ignored without doubt;

5 Patterns & Solutions

Performance bugs cannot all be modeled as rare events, features of them compare with traditional bugs:

1. They have similarities with traditional bugs;
2. They take longer time to be detected;
3. Developers cannot fight these bugs by themselves, they **cannot predict future workload or code changes to avoid bugs**. Research and tool support are needed here;

Some other features of performance bugs:

- **Location**: Three quarters of bugs are located **inside either an input-dependent loop or an input-event handler**; Half involve **I/Os or other time-consuming system calls**;

- **Correlation**: Skippalbe function root cause and the Change Condition bug-fix strategy are the most positivly correlated; Workload Issues bug-introducing reason is strongly correlated with the Change-A-Parameter bug-fix strategy; And Uncoordinated Functions root cause and the API Issues bug-introducing reason are the third most positively correlated pair.

5.1 Root Causes

Patterns:

1. **Uncoordinated Functions**: Inefficient function-call combinations are composed of efficient individual functions;
2. **Skippalbe Function**: Functions conduct unnecessary work given the calling context;
3. **Synchronization Issues**: Unnecessary synchronization that intensifies thread competition;
4. **Others**: Variety of reasons: wrong data structures, hardware architecture issues, high level design/algorithm issues

5.2 Introduced

- **Workload Mismatch**: Developers' workload understanding does not match with really: ① The input paradigm could shift after code implementation; ② Software workload has become much more diverse and complex than before;
- **API Misunderstanding**: Developers misunderstand the performance feature of certain functions: ① Performance of certain function is sensitive to the value of a particular parameter; ② Irrelevant task's performance is hurt by another function's invokation; ③ Some APIs have few or no documentations;
- **Changes**:Some bugs were not bug before: ① Workload shift makes some function implementation become inefficient; ② New hardware and feature become available to use.
- **Others**: Multiple reasons.

5.3 Exposed

Challenges for performance testing:

- **Always Active Bugs**: A *non-negligible* portion of performance bugs are almost always *active*, located at the start-up phase, shutdown phase or other places that are exercised by almost all inputs;
- **Input Feature & Scale Conditions**: Most performance bugs need input **with special features to manifest**, at which black box testing is not good; And **large-scale inputs** are need to find majority of performance bugs;

5.4 Fixed

Authors verified that most performance bugs can be fixed through **simple changes**.

1. Change a function-call sequence, referred to as *Change Call Sequence*;
2. *Change Condition*. Check carefully different input and condition;
3. *Change Parameter*;

6 Detection

I have a question here: how to define a performance bug, how to know the program works inefficiently here. The author try to use their patterns or features summarized before, Which I think is good but not complete.Some rules' applyingconditions are statically checkable, and some dynamically. And LLVM is used to get sufficient code sequence conditions and data-flow information.