Linhai Song - Research Statement

My research interests span the areas of systems, security, and software engineering. The goal of my research is to help developers build more efficient, reliable, and secure software systems.

My dissertation research centers around software performance. Naturally, everyone wants software to run fast. Slow and inefficient software can easily frustrate end users and cause economic loss, and it has already caused several highly publicized failures. My research philosophy is to view the software inefficiency problem from the perspective of combating *performance bugs*. Performance bugs are software implementation mistakes that can cause inefficient execution. Performance bugs cannot be optimized away by state-of-practice compilers. Many of them are overlooked by in-house testing and go on to affect end users, causing severe performance degradation and a huge waste of energy in the field. Performance bugs are becoming more critical, with the increasing complexity of modern software and workload, the meager increases of single-core hardware performance, and pressing energy concerns. It is urgent to combat performance bugs.

To fight performance bugs, my methodology is to investigate existing approaches originally designed for functional bugs and try to apply, adapt, and extend them for performance bugs. My experience spans different stages of combating performance bugs: real-world bug understanding, bug detection, failure diagnosis, and automated bug fixing. In particular, I conducted the first characteristics study on performance bugs, based on 110 bugs randomly collected from five representative open-source software suites [10]. I built a static performance bug detection tool suite by using extracted efficiency rules from fixed performance bugs [10] and helped build a dynamic performance bug detection tool for inefficient nested loops [9]. I explored how to apply statistical debugging to performance failure diagnosis [8], and designed a series of static-dynamic hybrid analysis routines to provide more detailed diagnosis information for inefficient loops [4]. I designed three source-to-source code transformations to automatically fix performance bugs in parallel applications, which offload computation to Intel Xeon Phi manycore coprocessors [7].

In addition to my work on software efficiency, I also worked on two software reliability projects as part of my doctoral research. In the first, I helped build a concurrency bug fixing system [11], which can automatically eliminate atomicity-violation bugs. In the second, I studied change histories of critical sections in open-source software [6] to provide a better understanding of synchronization challenges faced by real-world developers.

After graduation, I worked in the area of security, with a focus on applying big data techniques to analyze and predict security incidents. Using the data repository VirusTotal, which contains billions of real-world malware labeled by state-of-the-art anti-virus engines, I studied characteristics of real-world malware, modeled how influence propagates across different anti-virus vendors, and explored the feasibility of building machine learning malware detectors based only on hash values of files [3, 5]. The study results have shed light on directions for future research and are assisting both anti-virus vendors and normal users in their fight against malware.

My research work has already had industrial and academic impact. My two performance bug detection techniques [9, 10] found hundreds of previously unknown performance bugs in mature open-source software, many of which have already been confirmed and fixed by developers. My performance bug fixing project [7] was runner-up in the competition for best paper at MICRO-47 (2014) and has also led to an issued patent [1]. My concurrency bug fixing system [11] won the ACM SIGPLAN Research Highlights Award [2]. As of now, my publications have 415 citations in total.

1 Dissertation Research

My research efforts to address the software inefficiency problem cover the whole stack of software systems, from hardware to software, and all aspects of combating software bugs, from understanding to detecting, diagnosing, and fixing.

Real-world Performance Bug Understanding. Research on performance bugs, like that on functional bugs, should be guided by empirical studies. Poor understanding of performance bugs is one cause of today's performance bug problem. In order to improve the understanding of real-world performance bugs, I co-led what we believe to be the first empirical study of real-world performance bugs, based on 110 bugs randomly sampled from five open-source software suites [10]. Our study examined the four dimensions that cover the complete lifetime of performance bugs: their root causes, how they are introduced, how to expose them, and how to fix them. We found that: (1) the main root causes and fix strategies of performance bugs are highly correlated; (2) many performance bugs are introduced by workload mismatch and misunderstanding of APIs' performance features; and (3) around half of the studied performance bugs will only manifest under inputs with both special features and large scales. This empirical study can guide future research on performance bugs, and according to Google Scholar, it has already been cited 125 times since 2012. Our study has also motivated our own performance bug detection and performance failure diagnosis projects.

Static/Dynamic Performance Bug Detection. Our empirical study shows that both statically checkable efficiency rules and violations of these rules exist widely in software. Inspired by this finding, we manually inspected final patches of fixed performance bugs in our studied performance bug set, extracted efficiency rules from 25 bug patches, and implemented static checkers to detect rules' violations [10]. In total, our static checkers found 332 previously unknown performance bugs from the latest versions of Apache, Mozilla, and MySQL. We reported some of the identified bugs to developers. Of these, 77 have been confirmed by developers so far, and 15 reported bugs have already been fixed. Our empirical study also finds that 90% of performance bugs involve loops, and 50% of performance bugs involve at least two levels of loops. Motivated by this finding, I helped a fellow graduate student build a novel automated test oracle named Toddler [9], which can leverage the well-established testing process for functional bugs to identify performance bugs caused by inefficient nested loops. Using Toddler, we found 42 new bugs in six Java projects. Based on our bug reports, developers so far have fixed 21 of these bugs and confirmed 6 more as real bugs.

Performance Failure Diagnosis. Due to the preliminary tool support, many performance bugs escape in-house performance testing and manifest in front of end users. After users report performance bugs, developers need to diagnose them and fix them. Diagnosing user-reported performance failure is another key aspect of fighting performance bugs.

We first investigated the feasibility and design space to apply statistical debugging to performance failure diagnosis [8]. After studying 65 user-reported performance bugs in our bug set, we found that the majority of performance bugs are observed through comparison, and many user-filed performance bug reports contain not only bad inputs, but also similar and good inputs. Statistical debugging is a natural fix for user-reported performance bugs. We evaluated three types of widely used predicates and two representative statistical models. Our evaluation shows that branch predicate plus two statistical models can effectively diagnose user-reported performance failure. The basic model can help diagnose performance failure caused by wrong branch decision, and the ΔLDA model can identify inefficient loops. We applied sampling to performance failure diagnosis. Our experimental results show that because of the unique nature of loop-related performance bugs, sampling can lower runtime overhead without sacrificing diagnosis latency, which is very different from functional failure diagnosis.

We then built LDoctor [4] through a two-step process to provide more fine-grained diagnosis information about inefficient loops. We first figured out a root-cause taxonomy for common inefficient loops through a comprehensive study of 45 inefficient loops. Our taxonomy contains two major categories: resultless and redundancy, as well as several subcategories. Guided by our taxonomy, we then designed a series of analysis for inefficient loops. Our analysis checks suspicious loops pointed out by statistical debugging, hybridizes static and dynamic analysis to balance accuracy and performance, and relies on sampling and other designed optimization to further reduce runtime overhead. Evaluation results under real-world inefficient loops show that LDoctor covers most root-cause subcategories, reports few false positives, and brings a low runtime overhead.

Performance Bug Fixing. Intel Xeon Phi coprocessors have recently been introduced as new members of the manycore family. Compared with GPUs, Xeon Phi coprocessors are easier to program, because they provide x86 compatibility and support many different programming models. To offload existing parallel loops, developers just need to add simple pragmas. However, our recent experience shows that this does not result in better performance, and that too many performance bugs are contained in offloaded parallel loops.

After careful investigation, we designed three source-to-source code transformations to automatically fix performance bugs contained in offloaded parallel loops [7]. The first transformation, data streaming, is designed to overlap data transfers between CPUs and coprocessors with computation on coprocessors to hide data transfer overhead and reuse memory on coprocessors. The second transformation, regularization, is designed to re-arrange the order of computation and regularize loops with irregular memory accesses. The last transformation is designed to support the efficient transfer of large pointer-based data structures between CPUs and coprocessors. The designed transformations benefited 9 out of 12 benchmarks in our experiments and improved performance by 1.16x - 52.21x. This work won MICRO-47 (2014) best paper runner-up.

2 Future Research

Going forward, I hope to leverage my areas of expertise to improve the performance, reliability, and security of various types of computer systems.

Performance. My intermediate research goal is to advance new aspects of combating performance bugs. For example, I would like to explore how to monitor algorithmic complexity during production runs. Online algorithmic monitoring can help developers understand real-world workloads, identify codes in superlinear complexity, and expose new performance optimization opportunities. Current methods of algorithmic profiling or monitoring incur more than 10x runtime overhead, which cannot be tolerated in production runs. I plan to build a lower-overhead tool that will collect runtime information and infer approximate complexity from deployed software. I would also like to provide tool support for performance testing. Our empirical study shows that almost half of studied performance bugs only manifest under inputs with both special features and large scales. Existing techniques, designed to generate inputs with good code coverage, focus only on special features. I plan to extend these existing input generation techniques with an emphasis on large scales. Another important challenge that arises during performance testing is how to automatically judge whether a performance bug has occurred. I plan to leverage existing dynamic performance bug detection techniques to build performance testing oracles.

Reliability. As big data is changing businesses across the board, more and more developers are working on using big data computing systems, like Hadoop and Spark, to process their massive amounts of data. My own experience in leveraging Spark to analyze VirusTotal's data has given me insight into the challenges of debugging big data applications. First, it is time-consuming to repeat a failure, and it may take developers several hours to receive notice of a runtime failure or an incorrect output. Second, without tool support it is almost impossible to identify failure-triggering inputs among millions of input records. Third, it is difficult to identify a failure's root cause, since the failure may propagate across different stages and across different nodes.

My longer term research direction is to provide tool support for debugging big data applications. I plan to start by studying real-world bugs in big data applications to figure out whether there are common bug patterns. Then I hope to build static bug detectors that can identify bugs following common patterns before big data applications are executed. To quickly identify the root causes of failures, I plan to build slicing tools that can analyze execution across different nodes, interactive breakpoint tools that can stop the execution for single records specified by developers, and log mining tools that can analyze large quantities of logs in big data applications. Finally, I hope to build automatic fixing tools that can patch bugs in big data applications. Since it is time-consuming to run a patched version from the very beginning after fixes have been applied, I hope my fixing tools can provide support to reuse the computation completed during failure runs.

Security. I plan to continue my research on security along two directions. One is to apply big data techniques to security, and the other is to detect vulnerabilities in devices from the Internet of Things (IoT) ecosystem.

Big security data provides opportunities to leverage data-centric methods to improve today's security techniques. For example, we can study what attackers have done on a large scale to figure out their strategies and predict what they will do in the future. In addition, by utilizing malware that has already been studied and labeled by security experts, we can learn how malware evolves and quickly detect emerging malware. We can also apply deep learning techniques on these labeled malware samples and build new malware detectors. It is also important to think about whether existing big data computing systems are suitable for security workload and to look for opportunities to improve current systems.

Vulnerability detection in the IoT ecosystem is more critical than ever before. Billions of connected devices have already been in use worldwide, and this number is growing rapidly. Hacked devices can leak sensitive information and can power denial of service attacks. Due to the limited computation resources on a single device, anti-virus software cannot be used to prevent attacks. To make things worse, the developers who build firmware for these devices often treat security as an afterthought. The need to quickly detect and patch vulnerabilities is increasing dramatically. To detect vulnerabilities, I plan to study mechanisms for various firmware and devices and to extend existing techniques, like symbolic execution and whitebox fuzzing. Once a new vulnerability has been identified, it is also important to quickly search affected devices. To that end, I plan to build an efficient bug search system for IoT products.

References

[1] Min Feng, Srimat Chakradhar, and **Linhai Song**. "Compiler Optimization for Many Integrated Core Processors". U.S. Patent No. 20150277877.

[2] SIGPLAN. "SIGPLAN Research Highlights Papers". `http://www.sigplan.org/Newsletters/CACM/Papers/`.

[3] **Linhai Song**, Ce Zhang, Yiying Zhang, Heqing Huang, and Wu Zhou. "Understanding Malwares and Anti-Virus Engines in the Real World". `http://songlh.github.io/eurosys_submission.pdf` (**Under Submission**).

[4] **Linhai Song** and Shan Lu. "Performance Diagnosis for Inefficient Loops". In *Proceedings of the 2017 International Conference on Software Engineering*, **ICSE'17**, May 2017.

[5] **Linhai Song**, Heqing Huang, Wu Zhou, Wenfei Wu, and Yiying Zhang. "Learning from Big Malwares". In *Proceedings of the 7th ACM SIGOPS Asia-Pacific Workshop on Systems*, **APSys'16**, 2016.

[6] Rui Gu, Guoliang Jin, **Linhai Song**, Linjie Zhu, and Shan Lu. "What Change History Tells Us About Thread Synchronization". In *Proceedings of the 2015 10th Joint Meeting on Foundations of Software Engineering*, **ESEC/FSE'15**, 2015.

[7] **Linhai Song**, Min Feng, Nishkam Ravi, Yi Yang, and Srimat Chakradhar. "COMP: Compiler Optimizations for Manycore Processors". In *Proceedings of the 47th Annual IEEE/ACM International Symposium on Microarchitecture*, **MICRO'14**, December 2014. (**Best Paper Runner Up**).

[8] **Linhai Song** and Shan Lu. "Statistical Debugging for Real-world Performance Problems". In *Proceedings of the 2014 ACM International Conference on Object Oriented Programming Systems Languages and Applications*, **OOPSLA'14**, October 2014.

[9] Adrian Nistor, **Linhai Song**, Darko Marinov, and Shan Lu. "Toddler: Detecting Performance Problems via Similar Memory-access Patterns". In *Proceedings of the 2013 International Conference on Software Engineering*, **ICSE'13**, May 2013.

[10] Guoliang Jin*, **Linhai Song***, Xiaoming Shi, Joel Scherpelz, and Shan Lu. "Understanding and Detecting Real-World Performance Bugs". In *Proceedings of the 33rd ACM SIGPLAN Conference on Programming Language Design and Implementation*, **PLDI'12**, June 2012. *: equal contributions.

[11] Guoliang Jin, **Linhai Song**, Wei Zhang, Shan Lu, and Ben Liblit. "Automated Atomicity-Violation Fixing". In *Proceedings of the 32nd ACM SIGPLAN Conference on Programming Language Design and Implementation*, **PLDI'11**, June 2011. (**SIGPLAN Research Highlights Award**).