Journal of Software Engineering and Applications, 2021, 14, 474-492
https://www.scirp.org/journal/jsea
ISSN Online: 1945-3124
ISSN Print: 1945-3116

A Study of Performance Testing in Configurable Software Systems

Xue Han

University of Southern Indiana, Evansville, Indiana, USA
Email: xhan@usi.edu

How to cite this paper: Han, X. (2021) A Study of Performance Testing in Configurable Software Systems. *Journal of Software Engineering and Applications*, **14**, 474-492.
https://doi.org/10.4236/jsea.2021.149028

Received: June 22, 2021
Accepted: September 10, 2021
Published: September 13, 2021

Abstract

Customizing applications through program configuration options has been proved by many open-source and commercial projects as one of the best practices in software engineering. However, traditional performance testing is not in synch with this industrial practice. Traditional performance testing techniques consider program inputs as the only external factor. It ignores the performance influence of configuration options. This study aims to stimulate research interest in performance testing in the context of configurable software systems by answering three research questions. That is, why it is necessary to conduct research in performance testing, what are the state-of-the-art techniques, and how do we conduct performance testing research in configurable software systems. In this study, we examine the unique characteristics and challenges of performance testing research in configurable software systems. We review and discuss research topics on the performance bug study, performance anti-patterns, program analysis, and performance testing. We share the research findings from the empirical study and outline the opening opportunities for new and advanced researchers to contribute to the research community.

Keywords

Configurable Software Systems, Performance Testing, Software Configuration, Performance Bug Study

1. Introduction

Software performance is an inseparable part of user experience. A natural question to ask is why performance problems have not been given much attention. Manufacturers make faster processors every year to make programs run faster [1]. To that end, faster machines may cover up the performance problems. Do faster machines always alleviate performance problems? Maybe not. For instance, powerful web servers may not deliver a better user experience if the performance bugs are caused by the client's browser. To make things worse, the lack of a clear definition of performance bugs and measures makes it easy to overlook performance problems. It reveals the multiple facets of software performance problems that affect both business and end-users.

Real-world performance problems can introduce an unresponsiveness experience to end-users [2]. It may cost businesses to lose customers. In certain cases, performance problems may even cause lawsuits [3]. Prior study shows that when performance bugs are reported to developers, it takes a long time to fix [4]. In general, performance bugs are harder to replicate [5] [6] and even more challenging to locate and fix [7] the root cause. Performance bugs may result from the lack of performance concerns, limited performance testing tools, and postponed performance testing in the software development life cycle (SDLC) [8].

Performance bugs may cause faster energy consumption. In recent decades, the computing power of consumer mobile devices is equal to or greater than personal desktop computers [9]. Unlike traditional devices, battery-powered devices are more sensitive to performance bugs. End-users are more likely to notice the performance bugs on mobile devices, especially when end-users' mobile devices consume an unexpectedly large amount of energy [10].

Modern large-scale software systems offer the flexibility to fine-tuning system behaviors through configuration options [4]. However, the configuration options are overly complicated. It is easy to make mistakes [6]. Many techniques have been proposed to conduct functional testing on configurations [11] [12]. However, it remains an open challenge to detect performance bugs through configuration options effectively.

In this study, we aim to answer the following research questions.

- Why do we conduct performance testing in configurable software systems? We examine the importance of performance testing and the uniqueness of performance testing research in configurable software systems.
- What are the state-of-the-art performance testing techniques? We examine and summarize techniques used in recent research.
- How to research performance testing for configurable software systems? We answer this question to show the road map for researchers interested in performance testing.

In this study, we make the following contributions.

- We conduct a literature study of performance testing in the context of configurable software systems.
- We share research study findings and suggest future performance testing research directions.
- We provide a research map to help researchers to navigate performance testing research topics in configurable software systems.

The rest of the paper is organized as follows. In Section 2, we study literature in performance testing related research. In Section 3, we discuss the research questions and results. In Section 4, we discuss the limitation of existing research and outline future works. Lastly, we conclude the study in Section 5.

2. Literature Study

We conduct a literature study of performance testing in configurable software systems to provide background for further discussion.

2.1. Performance Bug Study

Performance bug study provides an overview of the characteristics of performance bugs. It is necessary to understand performance bugs before attempting to conduct performance testing.

Performance Bugs Jing *et al.* [13] study 109 real-world performance bugs to provide insights on performance bug detection, fixing, avoidance, and testing. Table 1 lists web servers used as research subjects [14].

Performance Bug Report Study Unlike previous bug studies that focus on the characteristics of performance bugs, Song *et al.* [15] study the diagnosis process of user-reported performance bugs. They point out that more than half of the reported bugs get developers' attention from the noticeable differences between runs. This study also answers what bug reporters tend to include in their bug reports.

Non-Performance V.S. Performance Bugs Zaman *et al.* [16] randomly select 400 performance and non-performance web browser bug reports. They quantify the nature of performance bugs in comparison to the non-performance bugs. Table 2 lists web client subjects used in the prior research. Nistor *et al.* [17] study over 600 bugs from three open-source projects. They compare and contrast the difference of discovering, reporting, and fixing between performance bugs and non-performance bugs. Figure 1 shows some performance bug studies in the past decade.

Table 1. Web server subjects.

Subject	Language	Size	Description
Apache	C++	L	Hypertext transfer protocol server
Lighttpd	C	S	Lightweight web server
Tomcat	Java	L	Java Servlet, JSP Container

Project size is measured in line of code (LOC). S (<100 K); M (100 K - 500 K); L (>500 K).

Table 2. Web client subjects.

Subject	Language	Size	Description
Firefox	C++	L	Web Browser
Chrome	C++	L	Web Browser
Thunderbird	C++	L	Email Client
Mozilla	C++	L	Web Browser

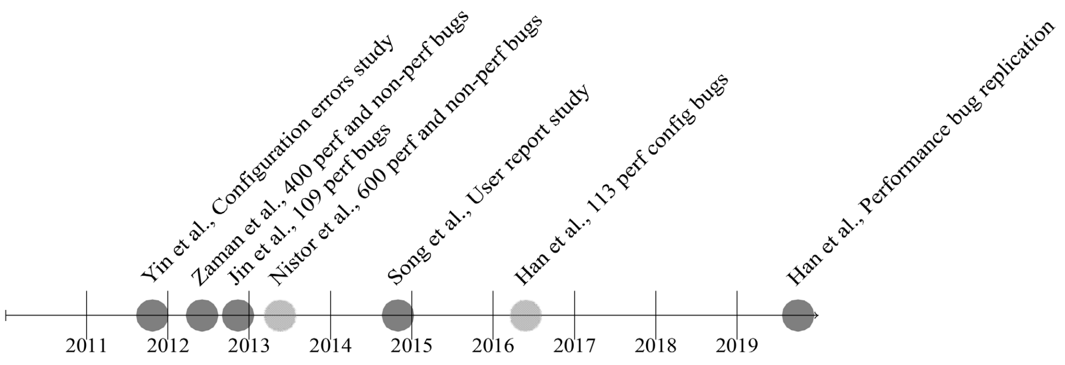

Figure 1. Performance bug studies in 2010-2019.

Findings 1: *Performance bug definition is not thoroughly defined in the research literature. Many research study relies on readers' intuition of what a performance bug is.*

It is necessary to define what a performance bug is to conduct performance bug studies[1]. However, defining performance bugs is not an easy task [18]. There are some performance bug definitions from prior research work. In terms of lines of code that need to fix, Jin *et al.* [13] define performance bugs as software defects where relatively simple source-code changes can significantly speed up the program. Zaman *et al.* [16] define performance bugs in terms of bug fixing. Performance bugs require not only experienced developers to fix but also take a longer time to fix. Foo *et al.* [19] define performance bugs in the context of regression testing-performance bugs are those defects that cause noticeable degradation of system performance when compared to previous release versions. Attariyan *et al.* [7] quantify performance bugs with performance metrics. Wert *et al.* [20] describe performance bugs as code defects that lead to low throughput, high response times, and high resource utilization.

Prior research fails to deliver a clear definition of what performance bugs are. Instead, such work relies on readers' intuition of what a performance bug is. The lack of an explicit description and a formal definition makes it harder to verify if a bug belongs to the performance bug category.

We define a software performance bug as code defects that complete a given task beyond the established level of resource utilization. The definition can be further tested with the following three criteria.

Assumption A performance bug is not a functional bug. It implies that the software system functions properly but may suffer from responsiveness. As such, bugs that cause systems to freeze or hang indefinitely are functional bugs first. ***Context*** A performance bug must be confirmed by developers. End-users may report a performance problem, but only developers have the authority to claim if the problem is indeed a performance bug. In many bug tracking systems, it is not unusual to see that some bug reporters confuse a performance improvement request with a performance bug.

[1]This work uses terms like performance bugs and performance problems. They are often not interchangeable.

Symptom A performance bug uses more resources (e.g., CPU and memory utilization or a time constraint) than the software specification [21]. Depending on the different types of subject programs, developers may choose the measures listed in **Table 3** to associate with performance bug symptoms. Performance is a subject matter. It is relevant to previous experience and is often judged based on previous experience by comparison. The most obvious observation of a typical performance bug is that the subject under test appears slower than expected.

Besides performance bugs, it is necessary to understand the configuration space to follow performance testing research in configurable software systems.

Configuration Space The configuration space has been studied in prior work [12] [22] [23]. Nair *et al.* [22] discuss the residual-based and rank-based approaches to sample the configuration space. Puoskari *et al.* [23] conduct a combinatorial testing case study in an industrial environment. Reisner *et al.* [12] conduct an empirical study of configuration space explosion in configurable software systems.

Misconfigurations Yin *et al.* [24] undertake a study to find characteristics of misconfiguration bugs in open-source and commercial systems. This work studies the system reactions (e.g., reliability, performance) caused by misconfiguration and reports the prevalence of different misconfiguration types.

2.2. Performance Anti-Patterns

Many performance analysis and testing tools [13] [25] [26] rely on matching specific performance bug patterns. In this section, we examine research focuses on software performance anti-patterns.

Coding Patterns Smith *et al.* [26] [27] [28] illustrate a set of anti-patterns that could lead to performance degradation. Unlike prior work that targets reported performance bugs, this study provides insights from software architecture design and programming language best practices to avoid performance bugs. The studied anti-pattens provide potential guidelines for rule-based [29] tools to

Table 3. Performance measures.

Performance Measure	DB	WS	GUI	Mobile
CPU Utilization	✓	✓	✓	✓
Memory Utilization	✓	✓	✓	✓
Cache Hit Rate	✓	✓		
I/O Utilization	✓	✓	✓	✓
Socket Utilization	✓	✓	✓	✓
Transactions	✓	✓		
Lock Contention Rates	✓	✓		
Response Time	✓	✓	✓	✓
Concurrent Request Rates	✓	✓		
Energy Consumption				✓

discover various performance bugs.

ORM Anti-Patterns Chen *et al.* [25] propose a framework to find performance anti-patterns in Object-Relational Mapping (ORM). By utilizing taint analysis, the framework identifies code path for data access and detect anti-pattern using data flow and rule-based approaches. Performance assessment is done by statistically evaluating the performance gain associated with each type of predefined anti-pattern before and after the code fix. **Table 4** lists database servers used as research subjects [4] [30] [31].

Anti-Pattern Detection Wert *et al.* [32] present a performance problem dianostics (PPD) method. This work combines search algorithms and decision tree algorithms to find known performance anti-patterns. Specifically, it provides a search hierarchy and a heuristic detection strategy to decide if performance anti-patterns exist in the system. Another approach to detecting performance problems is to utilize rule-based patterns. Jin *et al.* [13] synthesize a rule-based checker [29] [33] that utilizes the characteristics of their performance bug study to find potential performance problems (PPPs).

Findings 2: *Detecting performance anti-patterns depend on developers' expertise in a define-first-then-match approach. Many state-of-the-art tools are focusing on detecting rather than preventing performance anti-patterns in source code.*

Much research work relies on developers' expertise to prevent introducing performance anti-patterns into source code. However, we have not discovered much research on detecting anti-patterns in the development environment. Such techniques may prevent performance bugs from getting into the codebase, thus reducing the overall project cost [34].

2.3. Performance Analysis

Program analysis uses source code (static analysis) and code artifacts (dynamic analysis) to infer program properties such as correctness and performance. Static program analysis does not execute the project source code but requires source code to perform tasks like program flow analysis and impact analysis. Static analysis is language-specific. Popular static analysis tools include CodeSurfer [35], Clang [36], and FindBugs [37].

Dynamic program analysis, on the other hand, needs to run the subject program. Dynamic analysis involves code instrumentation and trace analysis. Although dynamic analysis incurs an execution overhead, it delivers much less

Table 4. Database server subjects.

Subject	Language	Size	Description
MySql	C	L	DBMS
PostgreSQL	C	L	DBMS
HBase	Java	L	Non-relational DBMS
Cassandra	Java	M	NoSQL DB

false-positive analysis results. Some popular dynamic instrumentation tools include Intel Pin [38], BTrace [39], ASM [40], and Soot [41]. Table 5 compares the technical differences between static and dynamic program analysis.

Loop Analysis Performance analysis uses both static and dynamic analysis. A few research work targets finding loops that lead to performance bottlenecks. Xiao *et al.* [42] present a delta inference technique (DeltaInference) for identifying workload-dependent performance bottlenecks. DeltaInference monitors the order of magnitude changes to detect performance bottlenecks. Nistor *et al.* [43] study over 100 performance bugs and identify four types of nested loops performance bugs. This work proposes an automated oracle (Toddler) for performance problems caused by repetitive memory-access patterns. Toddler uses the instruction pointer and call stack (IPCS) to determine if two IPCS-sequences are comparable and finds unnecessary computations in a loop. Table 6 lists the API subject for performance studies.

Idleness Analysis Enterprise-class applications have unique performance characteristics. For example, undesirable system idling may play a major role in slowing down system performance. Altman *et al.* [44] present a tool (WAIT) to find the root cause of system idling. WAIT is non-intrusive. It uses the sampling information readily available via Java Virtual Machines (JVM) and operating systems.

Researchers have designed multiple techniques [5] [7] to ease the process of

Table 5. Static v.s. dynamic program analysis.

Measure	Program Analysis	
	Static Analysis	**Dynamic Analysis**
Compilation Cost	Slow	Fast
Execution Overhead	Small	Large
False-Positive Rate	High	Low
Analysis Input	Source Code	Binary Code

Table 6. Library/API Subjects.

Subject	Language	Size	Description
Eclipse SWT	Java	M	The Standard widget toolkit
Hadoop	Java	L	Distributed processing library
Apache Collections	Java	S	Utility libraries
Guava	Java	M	Google core libraries
JFreeChart	Java	S	Chart Library
Lucene	Java	M	Search engine library
PDFBox	Java	S	PDF library
lbzip2	C	S	Block Compressor
OpenLDAP	C	L	LDAP libraries

diagnosing bugs and pinpoint the root cause of performance bugs.

Pinpoint Root Cause Attariyan *et al.* [7] present X-ray for diagnosing performance problems caused by configuration options in production environments. X-ray ranks a list of potential root cause configuration options based on their performance summarization. Dean *et al.* [5] present an online performance bug inference tool (PerfScope) to understand the occurrence of performance anomalies in cloud computing environments. PerfScope monitors and analyzes system call traces to identify inconsistent system call sequences.

Findings 3: *Choosing performance analysis strategies to detect performance problems depends on the software program types. Developers should balance factors such as false positive rate, execution cost, and the availability of source code before applying a specific performance analysis technique.*

2.4. Performance Testing

The goal of performance testing is to detect performance bugs in software systems. Many research focuses on performance test case generation [2] [45] [46] and performance regression testing [19] [47].

Test Generation Pradel *et al.* [2] design a technique (EventBreak) to find event pairs in the graphical user interface (GUI) applications where trigging one event would increase the execution time in the other event. Luo *et al.* [46] propose a black-box approach to learn rules from the execution traces. Such rules are used to find performance bottlenecks with feedback-directed software testing. Barna *et al.* [45] present a method to explore the workload space and pick a workload that leads to the worst performance. Nistor *et al.* [43] propose an automated oracle (Toddler) to detect performance problems. Unlike profilers which focus on a given metric and report the potential location where the hotspot resides purely based on ranked measurements, Toddler asserts performance when it finds unnecessary computations in the loop.

Performance Regression Regression testing is a crucial part of the continuous integration (CI) process. Many research tries to improve software quality through performance regression testing. Foo *et al.* [19] focus on detecting performance regression using historical data (logs files collected from heterogeneous testing environments) to build an ensemble performance prediction model. Unlike prior research, this approach challenges the traditional assumption that the test environment is consistent throughout the same organization. Huang *et al.* [47] propose a prioritized regression testing based on performance risk analysis (PRA) via static code change analysis. The PRA design involves a cost model to assess expenses associated with the committed change as well as an estimation of the frequency of the changed instruction to evaluate the risk level. In performance regression testing, it is important to setup the proper threshold using performance measurements. For instance, when performance is measured by benchmark tools, a tempting but false claim may be to say that if system performance does not slow down by 2% when compared to its previous version, we do not have a performance bug [48].

Findings 4: *Performance test input generation can achieve a high-level of automation whereas test oracles are still labor-intensive. Performance testing oracle research is falling behind.*

Table 7 summarizes the miscellaneous subjects studied in prior work.

3. Result

In this section, we answer the research questions and discuss the study results.

RQ1 Why do we conduct performance testing in configurable soft-ware systems? Testing is hard. Many developers are hesitant to do functional testing, let alone performance testing. However, performance testing is critical to the quality of software performance. Software systems that suffer from performance bugs could cost millions of dollars to fix [49].

Compared to the functional bugs, performance bugs are substantially more challenging to handle because they often manifest themselves with only large testing inputs and specific execution environments [43]. Plus, many existing performance testing approaches ignore configurations as a source of testing input.

Configurable software systems complicate performance testing. Prior study [17] shows that performance bugs in configurable software systems are more complex and take a longer time to fix. The sheer size of the configuration space makes the quality of software even harder to achieve.

Configuration-related performance bugs are prevalent. Prior work [4] finds that more than half of the performance bugs (59%) are due to configurations. We need more automated performance testing methods to handle the ever-increasing prevalence of configuration-related performance bugs in complicated configurable software systems. Performance testing research can improve the overall quality of software products and free developers from writing test cases manually.

RQ2 What are the state-of-the-art performance testing techniques? Research in performance testing is evolving at a fast speed. We examine the latest

Table 7. Misc. subjects.

Subject	Language	Size	Description
GCC Suite	C++	L	The GNU compiler collection
Notepad++	C++	M	Text editor
7-Zip	C++	S	File manager
Eclipse JDT	Java	M	IDE
Ant	Java	M	Build automation tool
Groovy	Java	M	Dynamic language
JMeter	Java	S	Load testing tool
Solr	Java	M	Search engine
Randoop	Java	L	Unit test generation

techniques applied to performance testing-related research. As discussed in Section 2, traditional program analysis techniques such as slicing [50], instrumentation [51], and taint analysis [52] have been widely adopted. Recent techniques take advantage of the advances in data mining and machine learning methods. We summarize performance testing techniques in **Table 8**. The "Technique" column shows the core algorithms or techniques used for the tool. For instance, DeltaInfer uses both instrumentation and machine learning techniques. Because instrumentation is used to extract code features to learn a prediction model with machine learning techniques, we only list "machine learning" as its core algorithm in the "Technique" column. The "Limitation" column outlines the constraints and underlining limitations associated with the technique. For instance, model checking techniques such as symbolic execution are notorious for scalability.

RQ3 How to research performance testing for configurable software systems? As discussed in section 2, performance testing in configurable software systems research involves much more than just testing. Researchers need to have a background in various research areas. Researchers can get first-hand insights by studying projects that have performance bugs. One way to understand performance bugs is through studying open-source projects with bug repositories [53] (*i.e.*, issue trackers). **Table 9** lists a few popular open-source projects with publicly accessible bug repositories. Researchers can gain insights into the root causes [13] of performance bugs by examining the bug reports. It lays out the ground for performance analysis. Performance analysis focuses on extracting the software performance properties. One type of performance analysis is based on performance anti-pattern matching. Researchers have broadly applied program analysis techniques to understand performance problems in the source code. Performance testing may use performance analysis techniques to generate intelligent inputs. The goal of performance testing is to detect performance bugs. Unlike performance analysis, performance testing is often part of the CI. **Figure 2** provides a quick reference to research topics that are relevant to performance testing in configurable software systems.

4. Discussion

Performance testing in configurable software systems is in its infant stage. There

Table 8. State-of-the-art performance analysis and testing techniques.

Name	Technique	Limitation
RuleChecker	Data Mining	High false negatives
X-ray	Taint Analysis	Low path coverage
Toddler	Instrumentation	High execution overhead
DeltaInfer	Machine Learning	Limited model interpretability
iTree	Clustering	Tuning hyper-parameters
Mantis	Static Slicing	Sensitivity to inputs

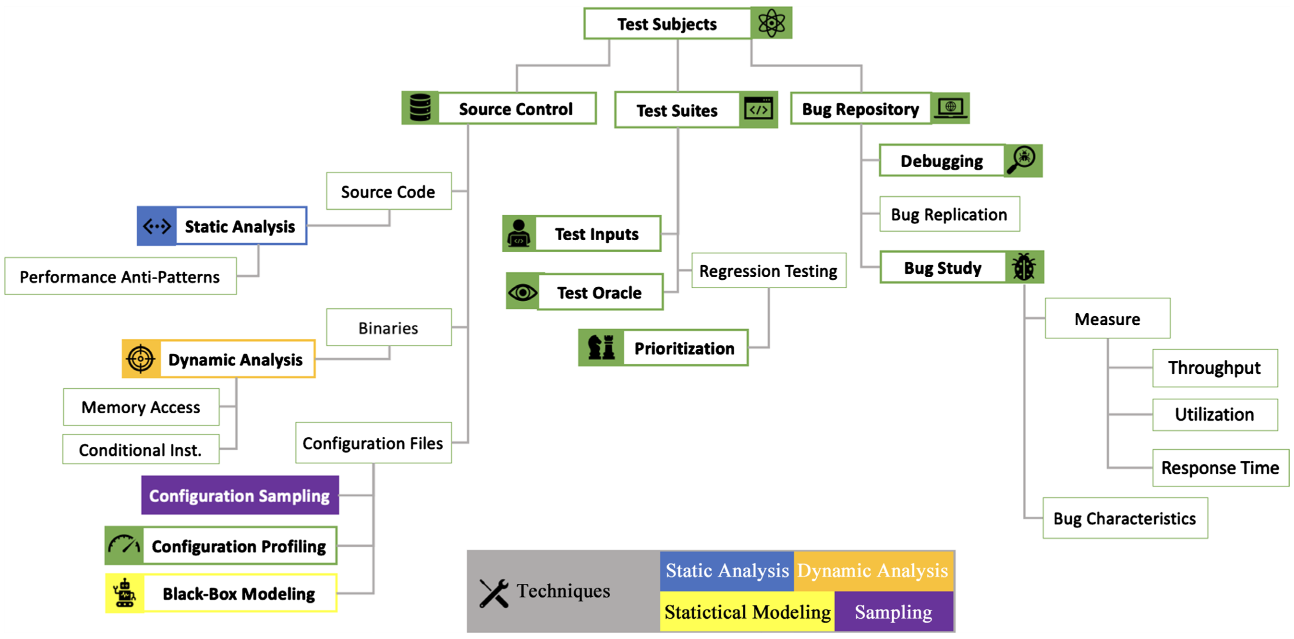

Figure 2. Performance testing research tree.

Table 9. Open-source bug repository.

Project	Project Bug Repository
Chromium	https://bugs.chromium.org/p/chromium/issues/list
Lucene	https://issues.apache.org/jira/browse/LUCENE
ASF	http://issues.apache.org/bugzilla
Mozilla	https://bugzilla.mozilla.org
MySQL	http://bugs.mysql.com

are many research topics to discover and explore. In this section, we discuss the limitation of the existing approaches and provide some future research directions.

4.1. Performance Bugs

The bug report is essential for developers to keep track of the status of the software system. Reporting performance bugs involve many manual efforts from both end-users and developers. In this section, we discuss how research in bug report automation can improve the efficiency in software development and the quality of software products.

Bug Report Han *et al.* [6] conduct an empirical study to reproduce performance bugs from bug reports. In their study, only less than 20% of the performance bugs are reproduced successfully. The quality of the performance bug reports leaves much to be desired. Research in the bug repository platform could help to improve the quality of the bug report. For example, the bug repository should check the completeness of the bug report. Bug reports need to include both the steps to reproduce the performance bug and the observable bug symp-

toms. Another research area is to study how to encourage end-users to report a performance problem. Many performance bugs are triggered under specific input, and they are difficult to reproduce. End-users may have observed the symptom and noticed the performance bug for a short time. But they may lack the expertise or tools to recreate and report those performance bugs accurately.
Bug Labeling When the bug study starts, the first step is to choose the correct categories of bugs. To categorize different bug types, much effort relies on manual inspections [4]. The bug study requires a large sampling size of bug reports to make a statistically significant conclusion that is not subject to generalization concerns. Thus, it is desirable to design approaches that can automate bug categorization. Some bug repositories (e.g., Apache) provide a tag field to ask bug reporters to specify the bug categorization. However, labeling a bug report is not required. There is no guarantee that bug reporters will provide the proper categorization of the bug. One solution may be to utilize supervised and unsupervised machine learning techniques to classify the bug report, hence applying automated bug labeling.

Performance Bug Study Performance bug studies are conducted by researchers manually. It is time-consuming. For instance, Han *et al.* [6] spend 800+ hours studying and reproducing performance bugs. Performance bug studies may become tedious over time and hurt the researcher's productivity. As artificial intelligence technologies mature, researchers may utilize natural language processing and understanding to find a way to automate and streamline the bug study process.

4.2. Configurations

Research in software configurations addresses concerns like prioritization and sampling in the configuration space.

Space Exploration For large-scale configurable software systems, it is not uncommon to have hundreds of configuration options. The configuration space is too large to explore extensively. Reisner *et al.* [12] conduct an empirical study on software configuration space explosion. With performance concerns, researchers must answer two questions to explore the performance configuration space efficiently. The first question is about which configuration options are performance influential. The second question is about what configuration option values may trigger a performance bug. Existing research work [14] [54] tries to provide solutions for those questions, but the configuration space exploration still suffers from scalability and solving constraints among multiple options.

Configuration Selection How to effectively get the configurations relevant to performance from a large space remains a research challenge. Applications of such techniques include selecting performance-sensitive configurations from a configuration space. Song *et al.* [55] propose iTree, a tool to achieve high code coverage configurations with decision trees. Unlike the combinatorial interaction testing (CIT), iTree is not restricted to the lower dimensions (two-way/

three-way samplings in CIT terms). Researchers may utilize similar strategies to find configurations that maximize performance-sensitive code coverage. The next step involves identifying an effective configuration option value range since performance bugs are triggered under a value range [42]. Most existing approaches restrict the configuration option value types and investigate only a low degree of configuration option interactions. More research efforts are also needed to associate what configuration options may lead to a performance bug under different scenarios. In prior work [4], researchers find that system-related configurations can cause significant performance degradations. As the cloud platform (e.g., AWS) and virtualization (e.g., VMware, Hyper-V) become prevalent, we need more studies to evaluate the performance influence of system-related configurations on those environments.

4.3. Performance Testing

Performance testing has been an important research topic. In this section, we discuss the desirable research areas that can advance performance testing research.

Performance Specifications Documentation like the software requirement is not always available. Performance specifications [56] may be even harder to find. Ideally, the performance specification should be used to establish a base-line for performance measurements. The performance measurement baseline is necessary to construct performance testing oracles. One research direction is to design a procedure to establish a performance measurement baseline when the specification is unavailable. For example, we may recreate the specification from valuable release notes, design documentation, code comments, and source code. Researchers may find an approach to compile and synthesize available documentations to construct decent performance specifications.

Performance Regression Testing Regression testing provides an approach to preserve existing functionality as the program evolves [57]. However, performance regression testing gets little attention. It is not unusual to see discussions in a bug report that a performance bug is introduced a few versions ago but only to surface in the bug report recently. The earlier that we can catch a performance bug, the cheaper it costs to fix the bug. We need better performance regression testing tools to detect performance bugs efficiently for fast-evolving large-scale applications. For example, when developers commit a code change, researchers may automate the process by analyzing the source code to determine if it is necessary to trigger the regression testing. It may potentially reduce the cost of executing expensive regression testing suites each time.

Performance Benchmarks Publicly accessible performance benchmarks [58] are necessary to evaluate performance techniques. Han *et al.* [6] discuss challenges to replicate performance bugs from the bug repository. For those research work evaluated with private projects [24], it is challenging to apply new techniques to the same projects due to security and privacy concerns. Most research work uses the same subject projects since there are only a few publicly available performance benchmarks.

Researchers need better subject program diversity in performance benchmarks. Much work is needed to collect and publish real-world performance benchmarks. Besides performance benchmarks, we may also need a public repository to host and share performance measurements. Yin *et al.* [24] study commercial systems and examine if the open-source project study findings apply to the commercial software. A public performance measurements repository may be necessary to establish a cross-project baseline for comparing performance among different projects in the same domain.

Bug Detection Nistor *et al.* [17] report that most (up to 57%) performance bugs are discovered with code reasoning. Code reasoning involves code understanding. Researchers may formulate the code reasoning problem as a machine learning problem. Another direction may be to utilize a record and replay method to employ code reasoning learned from one code block to apply to another code block in the same project.

Besides the automated approaches, researchers may find innovative ways to take advantage of crowdsourcing. For instance, some projects recruit crowdsource to mark labels for image processing through interactive games. Performance testing researchers may come up with creative ways to attract people to participate in detecting performance anti-patterns.

4.4. Performance Modeling

Performance models are widely used for understanding software performance and predict performance outputs. In this section, we discuss research directions utilizing modeling techniques in performance testing.

Performance Understanding Siegmund *et al.* [54] propose a black-box approach to build performance influence models for configurable software systems. The linear regression model is used to understand the performance influence of individual configuration options and their interactions. Han *et al.* [14] build location-level models to calculate the ranks of the performance influence of configuration options under different software usage scenarios. It is interesting to see how existing performance test generation techniques can leverage the performance understanding methods to generate effective test inputs.

Performance Prediction Kwon *et al.* [59] present a framework (Mantis) to predict the performance of Android applications. Mantis combines an offline and online approach to build a performance prediction model with executable program slices. Huang *et al.* [60] use sparse polynomial regression to predict software performance. Guo *et al.* [61] adopt a statistical learning approach to predict performance in configurable software systems.

Most performance models use a considerable amount of time to build. For projects that evolve fast, rebuilding a performance model from scratch every time is not practical. It is especially true when part of the workflow might depend on the performance model (e.g., performance models may be used for performance testing oracles). One potential research area may focus on how to build incremental performance models to reduce the cost.

5. Conclusion

We conduct a study of performance testing research in configurable software systems. Through this study, we address three essential research questions regarding performance testing in configurable software systems. The study covers a wide range of research topics in performance testing to guide new and advanced researchers. We share our study findings, discuss the boundary and limitations of the state-of-the-art techniques, and suggest future directions in performance testing research.

Acknowledgements

This work is funded by the University of Southern Indiana 2020 Faculty Research and Creative Work Grant.

Conflicts of Interest

The author declares no conflicts of interest regarding the publication of this paper.

References

[1] Webb, G.K. (2004) Predicting Processor Performance. *Issues in Information Systems*, **5**, 340-346.

[2] Pradel, M., Schuh, P., Necula, G. and Sen, K. (2014) EventBreak: Analyzing the Responsiveness of User Interfaces through Performance-Guided Test Generation. *ACM SIGPLAN Notices*, **49**, 33-47. https://doi.org/10.1145/2714064.2660233

[3] Hollister, S. (2020) Apple Will Pay $113 Million for Throttling Older iPhones in New "Batterygate" Settlement. https://www.theverge.com/2020/11/18/21573710/apple-battery-gate-throttle-iphones-settlement-amount

[4] Han, X. and Yu, T. (2016) An Empirical Study on Performance Bugs for Highly Configurable Software Systems. *Proceedings of the 10th ACM/IEEE International Symposium on Empirical Software Engineering and Measurement*, Ciudad Real, September 2016, Article No. 23. https://doi.org/10.1145/2961111.2962602

[5] Dean, D.J., Nguyen, H., Gu, X., Zhang, H., Rhee, J., Arora, N. and Jiang, G. (2014) Perfscope: Practical Online Server Performance Bug Inference in Production Cloud Computing Infrastructures. *Proceedings of the ACM Symposium on Cloud Computing*, Seattle, 3-5 November 2014, 1-13. https://doi.org/10.1145/2670979.2670987

[6] Han, X., Carroll, D. and Yu, T. (2019) Reproducing Performance Bug Reports in Server Applications: The Researchers' Experiences. *Journal of Systems and Software*, **156**, 268-282. https://doi.org/10.1016/j.jss.2019.06.100

[7] Attariyan, M., Chow, M. and Flinn, J. (2012) X-Ray: Automating Root-Cause Diagnosis of Performance Anomalies in Production Software. *10th USENIX Symposium on Operating Systems Design and Implementation*, Hollywood, 7-10 October

2012, 307-320.

[8] Molyneaux, I. (2009) The Art of Application Performance Testing: Help for Programmers and Quality Assurance. O'Reilly Media, Inc., Sebastopol.

[9] Samsung for Business (2020) Your Phone Is Now More Powerful than Your PC. https://insights.samsung.com/2020/08/07/your-phone-is-now-more-powerful-than-your-pc-2/

[10] Pathak, A., Hu, Y.C. and Zhang, M. (2011) Bootstrapping Energy Debugging on Smartphones: A First Look at Energy Bugs in Mobile devices. *Proceedings of the 10th ACM Workshop on Hot Topics in Networks*, Cambridge, 14-15 November 2011, Article No. 5. https://doi.org/10.1145/2070562.2070567

[11] Computer Security Resource Center (2016) Automated Combinatorial Testing for Software. http://csrc.nist.gov/groups/SNS/acts/index.html.

[12] Reisner, E., Song, C., Ma, K.-K., Foster, J.S. and Porter, A. (2010) Using Symbolic Evaluation to Understand Behavior in Configurable Software Systems. 2010 *ACM/IEEE 32nd International Conference on Software Engineering*, Cape Town, 1-8 May 2010, 445-454. https://doi.org/10.1145/1806799.1806864

[13] Jin, G., Song, L., Shi, X., Scherpelz, J. and Lu, S. (2012) Understanding and Detecting Real-World Performance Bugs. *ACM SIGPLAN Notices*, **47**, 77-88. https://doi.org/10.1145/2345156.2254075

[14] Han, X., Yu, T. and Pradel, M. (2021) Confprof: White-Box Performance Profiling of Configuration Options. *Proceedings of the ACM/SPEC International Conference on Performance Engineering*, Virtual Event, 19-23 April 2021, 1-8. https://doi.org/10.1145/3427921.3450255

[15] Song, L. and Lu, S. (2014) Statistical Debugging for Real-World Performance Problems. *Proceedings of the 2014 ACM International Conference on Object Oriented Programming Systems Languages & Applications*, Portland, 20-24 October 2014, 561-578. https://doi.org/10.1145/2660193.2660234

[16] Zaman, S., Adams, B. and Hassan, A.E. (2012) A Qualitative Study on Performance Bugs. 2012 *9th IEEE Working Conference on Mining Software Repositories (MSR)*, Zurich, 2-3 June 2012, 199-208. https://doi.org/10.1109/MSR.2012.6224281

[17] Nistor, A., Jiang, T. and Tan, L. (2013) Discovering, Reporting, and Fixing Performance Bugs. 2013 *10th Working Conference on Mining Software Repositories (MSR)*, San Francisco, 18-19 May 2013, 237-246. https://doi.org/10.1109/MSR.2013.6624035

[18] Koenig, A. (2013) Performance Bugs: Not Just Hard to Detect, but Hard to Define. http://www.drdobbs.com/cpp/performance-bugs-not-just-hard-to-detect/240164448

[19] Foo, K.C., Jiang, Z.M.J., Adams, B., Hassan, A.E., Zou, Y. and Flora, P. (2015) An Industrial Case Study on the Automated Detection of Performance Regressions in Heterogeneous Environments. 2015 *IEEE/ACM 37th IEEE International Conference on Software Engineering*, Florence, 16-24 May 2015, 159-168. https://doi.org/10.1109/ICSE.2015.144

[20] Wert, A., Happe, J. and Happe, L. (2013) Supporting Swift Reaction: Automatically Uncovering Performance Problems by Systematic Experiments. 2013 *35th International Conference on Software Engineering (ICSE)*, San Francisco, 18-26 May 2013, 552-561. https://doi.org/10.1109/ICSE.2013.6606601

[21] Alagar, V.S. and Periyasamy, K. (2011) Specification of Software Systems. Springer Science & Business Media, London. https://doi.org/10.1007/978-0-85729-277-3

[22] Nair, V., Menzies, T., Siegmund, N. and Apel, S. (2017) Using Bad Learners to Find Good Configurations. *Proceedings of the* 2017 11*th Joint Meeting on Foundations of Software Engineering*, Paderborn, 4-8 September 2017, 257-267. https://doi.org/10.1145/3106237.3106238

[23] Puoskari, E., Vos, T.E.J., Condori-Fernandez, N. and Kruse, P.M. (2013) Evaluating Applicability of Combinatorial Testing in an Industrial Environment: A Case Study. *Proceedings of the* 2013 *International Workshop on Joining AcadeMiA and Industry Contributions to testing Automation*, Lugano, 15 July 2013, 7-12. https://doi.org/10.1145/2489280.2489287

[24] Yin, Z., Ma, X., Zheng, J., Zhou, Y., Bairavasundaram, L.N. and Pasupathy, S. (2011) An Empirical Study on Configuration Errors in Commercial and Open Source Systems. *Proceedings of the* 23*rd ACM Symposium on Operating Systems Principles*, Cascais, 23-26 October 2011, 159-172. https://doi.org/10.1145/2043556.2043572

[25] Chen, T.-H., Shang, W., Jiang, Z.M., Hassan, A.E., Nasser, M. and Flora, P. (2014) Detecting Performance Anti-Patterns for Applications Developed Using Object-Relational Mapping. *Proceedings of the* 36*th International Conference on Software Engineering*, Hyderabad, 31 May-7 June 2014, 1001-1012. https://doi.org/10.1145/2568225.2568259

[26] Smith, C.U. and Williams, L.G. (2000) Software Performance Antipatterns. *Proceedings of the* 2*nd International Workshop on Software and Performance*, Ottawa, September 2000, 127-136. https://doi.org/10.1145/350391.350420

[27] Smith, C.U. and Williams, L.G. (2002) New Software Performance Antipatterns: More Ways to Shoot Yourself in the Foot. *Proceedings 28th International Conference Computer Measurement Group*, Reno, 8-13 December 2002, 667-674.

[28] Smith, C.U. and Williams, L.G. (2003) More New Software Performance Antipatterns: Even More Ways to Shoot Yourself in the Foot. 29*th International Computer Measurement Group Conference*, Dallas, 7-12 December 2003, 717-725.

[29] Li, Z. and Zhou, Y. (2005) PR-Miner: Automatically Extracting Implicit Programming Rules and Detecting Violations in Large Software Code. *ACM SIGSOFT Software Engineering Notes*, **30**, 306-315. https://doi.org/10.1145/1095430.1081755

[30] Han, X., Yu, T. and Lo, D. (2018) Perflearner: Learning from Bug Reports to Understand and Generate Performance Test Frames. 2018 33*rd IEEE*/*ACM International Conference on Automated Software Engineering* (*ASE*), Montpellier, 3-7 September 2018, 17-28. https://doi.org/10.1145/3238147.3238204

[31] Yu, T., Wen, W., Han, X. and Hayes, J.H. (2016) Predicting Testability of Concurrent Programs. 2016 *IEEE International Conference on Software Testing*, *Verification and Validation* (*ICST*), Chicago, 11-15 April 2016, 168-179. https://doi.org/10.1109/ICST.2016.39

[32] Wert, A. (2013) Performance Problem Diagnostics by Systematic Experimentation. *Proceedings of the* 18*th International Doctoral Symposium on Components and Architecture*, Vancouver, 17 June 2013, 1-6. https://doi.org/10.1145/2465498.2465499

[33] Hovemeyer, D. and Pugh, W. (2004) Finding Bugs Is Easy. *ACM SIGPLAN Notices*, **39**, 92-106. https://doi.org/10.1145/1052883.1052895

[34] Boehm, B.W. (1984) Software Engineering Economics. *IEEE Transactions on Software Engineering*, **SE-10**, 4-21. https://doi.org/10.1109/TSE.1984.5010193

[35] (2016) Codesurfer, a Code Browser That Understands Pointers, Indirect Function Calls, and whole-Program Effects.

https://www.grammatech.com/products/codesurfer

[36] (2016) Clang Static Analyzer. http://clang-analyzer.llvm.org/

[37] FindBugs (2016). http://findbugs.sourceforge.net/

[38] Levi, O. (2021) Pin—A Dynamic Binary Instrumentation Tool. https://software.intel.com/en-us/articles/pin-a-dynamic-binary-instrumentation-tool

[39] (2016) Btrace. https://kenai.com/projects/btrace

[40] (2016) ASM 4 Guide. http://download.forge.objectweb.org/asm/asm4-guide.pdf

[41] (2016) Soot. https://sable.github.io/soot/

[42] Xiao, X., Han, S., Zhang, D. and Xie, T. (2013) Context-Sensitive Delta Inference for Identifying Workload-Dependent Performance Bottlenecks. *Proceedings of the* 2013 *International Symposium on Software Testing and Analysis*, Lugano, 15-20 July 2013, 90-100. https://doi.org/10.1145/2483760.2483784

[43] Nistor, A., Song, L., Marinov, D. and Lu, S. (2013) Toddler: Detecting Performance Problems via Similar Memory-Access Patterns. 2013 35*th International Conference on Software Engineering* (*ICSE*), San Francisco, 18-26 May 2013, 562-571. https://doi.org/10.1109/ICSE.2013.6606602

[44] Altman, E., Arnold, M., Fink, S. and Mitchell, N. (2010) Performance Analysis of idle Programs. *Proceedings of the ACM International Conference on Object Oriented Programming Systems Languages and Applications*, Reno, 17-21 October 2010, 739-753. https://doi.org/10.1145/1869459.1869519

[45] Barna, C., Litoiu, M. and Ghanbari, H. (2011) Model-Based Performance Testing (Nier Track). *Proceedings of the* 33*rd International Conference on Software Engineering*, Waikiki, May 2011, 872-875. https://doi.org/10.1145/1985793.1985930

[46] Luo, Q., Nair, A., Grechanik, M. and Poshyvanyk, D. (2016) FOREPOST: Finding Performance Problems Automatically with Feedback-Directed Learning Software Testing. *Proceedings of the* 38*th International Conference on Software Engineering Companion*, Austin, 14-22 May 2016 593-596. https://doi.org/10.1145/2889160.2889164

[47] Huang, P., Ma, X., Shen, D. and Zhou, Y. (2014) Performance Regression Testing Target Prioritization via Performance Risk Analysis. *Proceedings of the* 36*th International Conference on Software Engineering*, Hyderabad, 31 May 2014-7 June 2014, 60-71. https://doi.org/10.1145/2568225.2568232

[48] Smaalders, B. (2006) Performance Anti-Patterns. *Queue*, **4**, 44-50. https://doi.org/10.1145/1117389.1117403

[49] Anthopoulos, L., Reddick, C.G., Giannakidou, I. and Mavridis, N. (2016) Why E-Government Projects Fail? An Analysis of the Healthcare.gov Website. *Government Information Quarterly*, **33**, 161-173. https://doi.org/10.1016/j.giq.2015.07.003

[50] Weiser, M. (1984) Program Slicing. *IEEE Transactions on Software Engineering*, **SE-10**, 352-357. https://doi.org/10.1109/TSE.1984.5010248

[51] Luk, C.-K., Cohn, R., Muth, R., Patil, H., Klauser, A., Lowney, G., Wallace, S., Reddi, V.J. and Hazelwood, K. (2005) Pin: Building Customized Program Analysis Tools with Dynamic Instrumentation. *ACM SIGPLAN Notices*, **40**, 190-200. https://doi.org/10.1145/1064978.1065034

[52] Schwartz, E.J., Avgerinos, T. and Brumley, D. (2010) All You Ever Wanted to Know about Dynamic Taint Analysis and Forward Symbolic Execution (but Might Have Been Afraid to Ask). 2010 *IEEE Symposium on Security and Privacy*, Oakland,

16-19 May 2010, 317-331. https://doi.org/10.1109/SP.2010.26

[53] Anvik, J., Hiew, L. and Murphy, G.C. (2006) Who Should Fix This Bug? *Proceedings of the 28th International Conference on Software Engineering*, Shanghai, 20-28 May 2006, 361-370. https://doi.org/10.1145/1134285.1134336

[54] Siegmund, N., Grebhahn, A., Apel, S. and Kästner, C. (2015) Performance-Influence Models For highly Configurable Systems. *Proceedings of the* 2015 10*th Joint Meeting on Foundations of Software Engineering*, Bergamo, 30 August-4 September 2015, 284-294. https://doi.org/10.1145/2786805.2786845

[55] Song, C., Porter, A. and Foster, J.S. (2014) Itree: Efficiently Discovering High-Coverage Configurations Using Interaction trees. *IEEE Transactions on Software Engineering*, **40**, 251-265. https://doi.org/10.1109/TSE.2013.55

[56] Sitaraman, M., Kulczycki, G., Krone, J., Ogden, W.F. and Reddy, A.N. (2001) Performance Specification of software Components. *ACM SIGSOFT Software Engineering Notes*, **26**, 3-10. https://doi.org/10.1145/379377.375223

[57] Wong, W.E., Horgan, J.R., London, S. and Agrawal, H. (1997) A Study of Effective Regression Testing in Practice. *Proceedings of the* 8*th International Symposium on Software Reliability Engineering*, Albuquerque, 2-5 November 1997, 264-274. https://doi.org/10.1109/ISSRE.1997.630875

[58] Dixit, K.M. (1991) The Spec Benchmarks. *Parallel Computing*, **17**, 1195-1209. https://doi.org/10.1016/S0167-8191(05)80033-X

[59] Kwon, Y., Lee, S., Yi, H., Kwon, D., Yang, S., Chun, B.-G., Huang, L., Maniatis, P., Naik, M. and Paek, Y. (2013) Mantis: Automatic Performance Prediction for Smartphone Applications. 2013 *USENIX Annual Technical Conference*, San Jose, 26-28 June 2013, 297-308.

[60] Huang, L., Jia, J., Yu, B., Chun, B.-G., Maniatis, P. and Naik, M. (2010) Predicting Execution Time of Computer Programs Using Sparse Polynomial Regression. *Proceedings of the* 23*rd International Conference on Neural Information Processing Systems*, Vol. 1, Vancouver, 6-9 December 2010, 883-891.

[61] Guo, J., Czarnecki, K., Apel, S., Siegmund, N. and Wasowski, A. (2013) Variability-Aware Performance Prediction: A Statistical Learning Approach. 2013 28*th IEEE/ACM International Conference on Automated Software Engineering* (*ASE*), Silicon Valley, 11-15 November 2013, 301-311. https://doi.org/10.1109/ASE.2013.6693089