Ting Dai RESEARCH STATEMENT

My research interests are in the areas of *software systems*, *program analysis*, and *cloud computing*. The goal of my research is to make software systems and cloud infrastructures *reliable* and *secure*. My research has made a significant impact:

- leading to thousands of code improvements in both open-source and commercial cloud systems [10, 19, 23],
- being widely commercialized and integrated into business products [5–7, 19],
- being broadly quoted by both internal press [8, 16, 26] and external tech media [17, 18, 24].

Dissertation Research. My dissertation research has focused on *performance bugs* (such as software hang and performance slowdown) which are notoriously difficult to debug due to the lack of accurate diagnosis information. Performance bugs pose a cacophony of risks to software applications, poisoning cloud infrastructures, and wreaking havoc on system reliability and availability. Particularly in the production cloud environment, performance bugs can be triggered by corrupted data or unexpected control flows, affecting thousands of nodes, causing cascading failures and endless retries, freezing server systems, leading to catastrophic service outages [1–3]. Many existing approaches tackling performance bugs assume full-stack domain knowledge or available source code [10, 23], which however is often impractical for the production cloud. Therefore, the desire has never been stronger for an application-agnostic and bytecode-based approach to capture and correct performance bugs.

My approach to combat performance bugs is built on the systematic understanding of their root-cause functions and manifestation patterns [9]. Such knowledge and experience are cultivated and distilled as fundamental principals to guide performance checking and repair on application bytecode or runtime syscall traces to enable generic performance bug diagnosis [20, 21], data-corruption caused hang bug detection [23], and corruption or inter-process communication failure induced hang bug fixing [10]. In addition to performance bugs, I have also worked on *timeout bugs*, starting from a comprehensive taxonomy [22]. The study results have shed light on directions for the later timeout research, which specifically motivated my work on timeout bug detection and classification [12], misused timeout bug fixing [13], and missing timeout bug fixing [11].

Industry Research. My industry research centers around *security*, which is at an all-time high, gaining incomparable attention from the federal government to tech giants [4]. Enacting security is not only fostering a cybersecurity conscious environment, but also applying technical safeguards and security processes to detect and prioritize threats, risks, and vulnerabilities in early stages. I have built such security practices as risky script detection in infrastructure code [19] integrated with DevOps pipelines, and environment-aware risk quantification [7, 8] with vulnerability weaponization metric enhancement [6]. Furthermore, security lapses raise subsequent *compliance* issues, which is particularly on the rise in cloud environments, where countless cloud-based offerings are easily accessible but unregulated. Establishing checkable rules and guidelines to meet regulation purposes is a must, which inspired my work on policy-based compliance checking and remediation in multi-clouds [5], such as AWS and Azure. However, compliance is not a guarantee against data breaches. To preserve *data privacy*, security and compliance strategies should incorporate data encryption and authentication, amongst which, third-party auditors with security decisions are preferable. With such intention, I introduced privacy-preserving and auditability into blockchains at the same time [15].

1 DISSERTATION AND CURRENT RESEARCH

My thesis focuses on addressing performance bugs in cloud systems spanning different stages from diagnosing to detecting and fixing. First, I present Hytrace [21], a hybrid approach to diagnosing performance problems by combing generic rule based static analysis and runtime inference techniques. I observed that many performance problems that happen in the cloud are caused by data corruptions, which cannot be captured by generic approaches including Hytrace. To proactively detect such bugs, I built DScope [23], a static checking tool that identifies loops whose exit conditions can be affected by error-prone I/O operations. To correct performance problems, especially software hang bugs caused by data processing or inter-process communication failures, I developed HangFix [10], an automatic fixing tool, which extracts commonly seen hang bug patterns and provides corresponding patching strategies based on the identified patterns. My recent industry research work SecureCode [19] targets risky scripts in infrastructure code that cause not only performance issues but also reliability, security, and availability problems. I will now summarize each of these projects.

1.1 Diagnosing performance problems with Hytrace

Previous work on performance bugs can be broadly classified into static rule-based code analysis and dynamic trace-based inference. Static approaches achieve high code coverage but lose focus on specific anomaly occurred in a production run, which inevitably report excessive false alarms. In contrast, dynamic approaches can target the specific problem with runtime monitoring but suffer from false negatives without knowing program semantics.

To retain the strengths and alleviate the weakness of each individual scheme, I proposed a hybrid approach, Hytrace [21]. Hytrace leverages carefully designed Hytrace-static and Hytrace-dynamic analysis components that are complementary to each other. Hytrace-static favors the coverage over precision. It captures all the potential buggy functions who are vulnerable to the performance problems over a set of *generic* anti-patterns such as constant parameter function calls and uncovered branches. Hytrace-dynamic favors both the coverage and the "relaxed" precision. It identifies the buggy functions who have abnormal runtime practices from kernel-level syscall traces using machine learning anomaly detection algorithms. It then extends candidate list by adding k-hop caller functions with the insight that callers bear buggy semantics which manifest in their callees. Although each analysis component is prone to false positives, the intersection of the two leverages both program semantics and run-time behavior information, and hence can achieve much higher precision than pure-static or pure-dynamic techniques.

I tested Hytrace on 133 real performance bugs from 9 representative open-source cloud systems. Hytrace provides 100% coverage over all tested bugs with only reporting 6 false positive functions on average for the 14 reproducible bugs.

1.2 Detecting data corruption hang bugs with DScope

From my experience with Hytrace, I found that many tricky performance bugs are caused by unexpected data corruptions. Especially with the ever-evolving cloud computing and nonstop data explosion, data corruption induced hang bugs are becoming more prevalent and disastrous [3]. The corrupted data from hardware faults, software bugs, or human errors are catalyzed by internode interactions until they manifest in infinite loops, which makes the bugs difficult to be captured by Hytrace's generic static rules or leave evidence on syscall traces.

To detect such corruption-hang bugs, I built DScope [23]. DScope takes into consideration the heterogeneous origins of data corruptions and the homogeneous manifestation patterns of infinite loops and carries out a bottom-up analysis—from manifestation detection to root-cause categorization. Specifically, DScope takes an application bytecode as input, statically identifies infinite loops (i.e., manifestation) by checking whether their exit conditions are affected by external data through variegated error-prone I/O operations (i.e., root-cause). To achieve high detection accuracy, DScope performs loop bound and stride analysis to filer out those loops which can always exit regardless of the I/O operations. From DScope's detection results, I found that common corruption-hang bugs are caused by unexpected returned data, returned error code, or corrupted data content from direct or indirect data assignments or control flow changes.

I tested DScope on 9 popular cloud systems. DScope successfully detects 42 true bugs with 29 of them are newly discovered. DScope is fully automatic and generally applicable to cloud systems without requiring any predefined rules or application specific information.

1.3 Fixing hang bugs with HangFix

With the accumulating knowledge of diagnosing and detecting performance bugs, particularly hang bugs, I noticed the urgency and desire for an automated hang bug fixing tool. Moreover, carrying forward my previous hang bug detection work, i.e., DScope [23] and TScope [12], such fixing tool should and must cover two major hang bug types in the cloud—unexpected runtime data corruption induced infinite loops (detected by DScope) and inter-process communication failure caused blocking operations (detected by TScope).

To fulfill the aforementioned purposes, I built HangFix [10] to automatically generate patch code for hang bugs. HangFix takes a divide-and-conquer approach by first categorizes the hang bugs as infinite loops or blocking operations. For infinite loops, HangFix's patching strategy is to insert safe checking and exception statements inside the loop to break the hang, which captures any anomalies exposed during runtime execution and prevents them from falling into loop iterations indefinitely. As for blocking operations, HangFix isolates them from the main application execution using a callable or runnable thread with timeout settings. HangFix makes the best effort to borrow the values from existing application configurations for the newly introduced timeout variables. To further provide more accurate timeout values, my recent work TFix+ [11] presents a prediction-driven approach based on runtime resources and application workloads.

I tested HangFix on 237 real-world hang bugs from 10 cloud systems. HangFix's patching strategies can cover 76% of all tested bugs. HangFix successfully fixes 40 out of 42 reproducible bugs completely in seconds with 28 are previously unfixed. HangFix's patch introduces less than 1% performance overhead to the applications without bringing any other side effects.

1.4 Detecting risky infrastructure scripts with SecureCode

My recent work focuses on debugging infrastructure as code where embedded risky scripts introduce not only performance problems but also reliability, availability, and security issues. Unlike cloud systems armed with organized bug repositories and comprehensive debugging practices, software bugs and vulnerabilities in infrastructure scripts are overlooked by sysadmins who lack the same level of understanding and debugging support compared with software developers.

To help identify risky infrastructure scripts, I built SecureCode [19] particularly to harden sysadmins' debugging force. SecureCode takes an infrastructure code repository as input and automatically composes script contents by removing template renderings after identifying that the infrastructure code invokes script executions. After passing the composed scripts to script analyzers, SecureCode assigns the severity level and potential impact for each identified issue based on a pre-defined knowledge base. The knowledge base is built on a systematic understanding of the existing script analyzers' rulesets, including 345 rules from ShellCheck and 64 rules from PSScriptAnalyzer. SecureCode categorizes impacts based on the code patterns checked by each rule, and measures severity levels based on the infrastructure operation criticality extracted from operation names, automation file names, and comments.

I integrated SecureCode with the DevOps pipeline deployed in IBM cloud and tested it on 45 IBM Services community repositories. SecureCode successfully identifies 3,419 availability, reliability, performance, and security issues residing in the infrastructure code, within which 1,691 have high severity levels.

2 FUTURE RESEARCH

With the growth of tech innovation, the demand for correct, efficient, secure, and eco-friendly software systems increases. I am excited to continue working in the emerging cloud systems and infrastructures to address many fundamental challenges, build practical solutions, and make a broad impact.

2.1 Enforcing performance debugging in different languages

My work HangFix [10] and SecureCode [19] show that performance bugs in different languages have different patterns. In Java cloud systems, corruption-loops and blocking operations are pervasive, while in Shell scripts, inefficient redirections and useless `cat` and `echo` dominate the performance issues. Extending performance debugging effort into software systems and infrastructures in other languages is needed with the following aspects to focus on: 1) broadening language syntactic and program semantic rulesets to cover the majority of performance bugs; 2) proposing language-specific compilers or frameworks to support different types of static analysis, such as data-/control-flow analysis and intra-/inter-procedural analysis; and 3) extracting performance bug patterns in kernel levels excluding language specification and building a general checkable model.

Furthermore, the existing performance debugging force works in silos, lacking support for interactive inter-language debugging. For example, HangFix cannot analyze the underline blocking procedure implemented in hardware and OS platform-specific library code written in `C/C++` and assembly. My insight is that *compiler integration* can eliminate language barriers with coordinated analysis components working together to deliver a debugging pipeline. The integrated compilers can have their own rulesets conducting specific analysis with carefully designed interfaces (i.e., pre-/post-processing) transmitting intermediate analysis results.

2.2 Open-source software continuous debugging and fixing

A recent study [14] shows that 98% of 1,546 commercial codebases contain open-source code and 84% of them contain at least one public open-source vulnerability with an average of 158 per codebase. Continuously debugging and fixing dependent open-source packages in software supply chains to enhance system security and robustness is a necessity with the following capabilities: 1) monitoring the development environment and updating dependent open-source packages to obtain the latest functionality and security patches; 2) automating debugging force and patching strategies to security vulnerabilities; 3) encouraging all developers and users to contribute by reporting their identified bugs and proposing corresponding patches into a public shared repository. I aspire to step towards the open-source software continuous debugging and fixing.

2.3 Building energy-efficient systems

Reducing carbon emissions to protect the global ecosystem is on the go and should be considered when building cloud systems and infrastructures. My recent work GreenABR [25] tackles the problem of sustaining high performance with substantial energy efficiency by implementing it in video streaming. Our evaluation results show that GreenABR excels all competitors with up to 48% power savings and eight times perceptual quality improvement.

We should not leave energy consumption as an afterthought but weave it into the fabric of system design, development, deployment, and maintenance with the following efforts to concentrate on: 1) avoiding computationally expensive operations instead choosing cost-efficient counterparts; 2) addressing performance bugs such as software hang problems with 100% CPU usages; and 3) balancing the tradeoffs between performance boost and energy savings. Higher performance is not always acceptable with the exponentially increased energy consumption. With the rapid growth of AI and Edge computing, one-shot model fusion, model adaption, "edge-in" engineering, and other energy-efficient schemes will be prevalent.

REFERENCES

[1] 2015. AWS outage knocks Amazon, Netflix, Tinder and IMDb in MEGA data collapse. https://www.theregister.co.uk/2015/09/20/aws_database_outage/

[2] 2015. Irreversible Failures: Lessons from the DynamoDB Outage. http://blog.scalyr.com/2015/09/irreversible-failures-lessons-from-the-dynamodb-outage/

[3] 2017. What Lessons can be Learned from BA's Systems Outage? http://www.extraordinarymanagedservices.com/news/what-lessons-can-be-learned-from-bas-systems-outage/

[4] 2021. FACT SHEET: Biden Administration and Private Sector Leaders Announce Ambitious Initiatives to Bolster the Nation's Cybersecurity. https://www.whitehouse.gov/briefing-room/statements-releases/2021/08/25/fact-sheet-biden-administration-and-private-sector-leaders-announce-ambitious-initiatives-to-bolster-the-nations-cybersecurity

[5] 2021. IBM Cloud Pak for Watson AIOps. https://www.ibm.com/cloud/cloud-pak-for-watson-aiops

[6] 2021. IBM Data Risk Manager. https://www.ibm.com/products/data-risk-manager

[7] 2021. X-Force Red vulnerability management and scanning services. https://www.ibm.com/security/services/vulnerability-scanning

[8] Muhammed Fatih Bulut, **Ting Dai**, and Steve Ocepek. 2021. New service from IBM Research and X-Force Red makes vulnerability management more efficient. https://www.research.ibm.com/blog/cloud-resistance-strength

[9] Daniel Joseph Dean, Peipei Wang, Xiaohui Gu, William Enck, and Guoliang Jin. 2015. Automatic Server Hang Bug Diagnosis: Feasible Reality or Pipe Dream?. In *Proceedings of the 12th IEEE International Conference on Autonomic Computing (ICAC '15).* https://doi.org/10.1109/ICAC.2015.52

[10] Jingzhu He, **Ting Dai**, Xiaohui Gu, and Guoliang Jin. 2020. HangFix: Automatically Fixing Software Hang Bugs for Production Cloud Systems. In *Proceedings of the 11th ACM Symposium on Cloud Computing (SoCC '20).* https://doi.org/10.1145/3419111.3421288

[11] Jingzhu He, **Ting Dai**, and Xiaohui Gu. 2021. TFix+: Self-configuring Hybrid Timeout Bug Fixing for Cloud Systems. arXiv:2110.04101

[12] Jingzhu He, **Ting Dai**, and Xiaohui Gu. 2018. TScope: Automatic Timeout Bug Identification for Server Systems. In *Proceedings of the 15th IEEE International Conference on Autonomic Computing (ICAC '18).* https://doi.org/10.1109/ICAC.2018.00010

[13] Jingzhu He, **Ting Dai**, and Xiaohui Gu. 2019. TFix: Automatic Timeout Bug Fixing in Production Server Systems. In *Proceedings of the 39th IEEE International Conference on Distributed Computing Systems (ICDCS '19).* https://doi.org/10.1109/ICDCS.2019.00067

[14] Synopsys Inc. 2021. Open Source Security and Risk Analysis Report. https://www.synopsys.com/software-integrity/resources/analyst-reports/open-source-security-risk-analysis.html

[15] Hui Kang, **Ting Dai***, Nerla Jean-Louis, Shu Tao, and Xiaohui Gu. 2019. FabZK: Supporting Privacy-Preserving, Auditable Smart Contracts in Hyperledger Fabric. In *Proceedings of the 49th IEEE/IFIP International Conference on Dependable Systems and Networks (DSN '19).* https://doi.org/10.1109/DSN.2019.00061 (*Co-primary author).

[16] NC State News. 2020. Software Spots and Fixes Hang Bugs in Seconds, Rather Than Weeks. https://news.ncsu.edu/2020/10/hangfix/

[17] News8Plus. 2020. Software program spots and fixes grasp bugs in seconds, fairly than weeks. https://news8plus.com/software-spots-and-fixes-hang-bugs-in-seconds-rather-than-weeks/

[18] The Register. 2020. We won't leave you hanging any longer: Tool strips freeze-inducing bugs from Java bytecode while in production. https://www.theregister.com/2020/10/13/jave_hanging_bug/

[19] **Ting Dai**, Alexei Karve, Grzegorz Koper, and Sai Zeng. 2020. Automatically Detecting Risky Scripts in Infrastructure Code. In *Proceedings of the 11th ACM Symposium on Cloud Computing (SoCC '20).* https://doi.org/10.1145/3419111.3421303

[20] **Ting Dai**, Daniel Dean, Peipei Wang, Xiaohui Gu, and Shan Lu. 2017. Hytrace: a hybrid approach to performance bug diagnosis in production cloud infrastructures. In *Proceedings of the 8th ACM Symposium on Cloud Computing (SoCC '17).* https://doi.org/10.1145/3127479.3132562

[21] **Ting Dai**, Daniel Dean, Peipei Wang, Xiaohui Gu, and Shan Lu. 2019. Hytrace: A Hybrid Approach to Performance Bug Diagnosis in Production Cloud Infrastructures. *IEEE Transactions on Parallel and Distributed Systems (TPDS)* 30, 1 (2019). https://doi.org/10.1109/TPDS.2018.2858800

[22] **Ting Dai**, Jingzhu He, Xiaohui Gu, and Shan Lu. 2018. Understanding Real-World Timeout Problems in Cloud Server Systems. In *Proceedings of the 6th IEEE International Conference on Cloud Engineering (IC2E '18).* https://doi.org/10.1109/IC2E.2018.00022

[23] **Ting Dai**, Jingzhu He, Xiaohui Gu, Shan Lu, and Peipei Wang. 2018. DScope: Detecting Real-World Data Corruption Hang Bugs in Cloud Server Systems. In *Proceedings of the 9th ACM Symposium on Cloud Computing (SoCC '18).* https://doi.org/10.1145/3267809.3267844

[24] SD Times. 2020. Researchers develop HangFix to quickly resolve hang bugs. https://sdtimes.com/softwaredev/researchers-develop-hangfix-to-quickly-resolve-hang-bugs/

[25] Bekir Turkkan, **Ting Dai**, Adithya Raman, Tevfik Kosar, Changyou Chen, Fatih Bulut, Jaroslaw Zola, and Daby Sow. 2022. GreenABR: Energy-Aware Adaptive Bitrate Streaming with Deep Reinforcement Learning. In *Proceedings of the 17th European Conference on Computer Systems (EuroSys '22).* (In submission).

[26] Sai Zeng. 2020. Kubernetes-based Control Plane to Manage Risk and Compliance for Hybrid Cloud. https://www.ibm.com/blogs/research/2020/08/kubernetes-based-control-plane-to-manage-risk-and-compliance-for-the-cloud/