Research Statement

Tianyin Xu

Today, with cloud and datacenter systems scaling massively to deliver utility services, the impact and cost of system failures increase dramatically. For example, the failure of Amazon EC2 Web Services brought down more than 70 popular websites, including Reddit, Quora, and Foursquare [1]. Despite the wide adoption of fault-tolerance and recovery techniques, cloud and datacenter systems still experience failures constantly.

The goal of my research is to build *reliable* and *secure* computer systems that operate at cloud and datacenter scale. My Ph.D. work has focused on tackling one dominant cause of cloud and datacenter failures in the real world—*configuration errors*—which are notoriously fatal and hard to deal with. In cloud and datacenter systems, the same configuration error is often replicated to hundreds and thousands of nodes, which significantly enlarges the error impact and makes redundancy-based fault tolerance ineffective. Moreover, as configuration errors typically reside in persistent files or databases, they are resistant to rebooting-based recovery techniques. To make matters worse, even fault tolerance and recovery are often misconfigured in reality, and such configuration errors can impair the immune system of the entire cloud and datacenter, leading to catastrophic outages [1–3].

Most existing approaches attack configuration errors by equipping system administrators (sysadmins) with troubleshooting facilities [4] to help diagnose the failure's root causes [4]. Unfortunately, these break-and-fix approaches cannot prevent failures and outages in the first place. In fact, as demonstrated in my thesis work, many severe configuration errors that caused catastrophic failures can be proactively captured and corrected.

My approach. I wield a different approach to the problem—rather than requesting correct configurations and blaming sysadmins for misconfigurations, I believe the more fundamental and practical solution is to enable the systems to anticipate and defend against configuration errors. In this way, even if configuration errors are introduced in the field, the systems can report the errors timely and react to the errors gracefully to reduce failure damage.

I have built a number of enabling techniques for building reliable and secure systems in the face of misconfigurations, one of the most difficult problems that remain in system reliability for decades. Compared with software bugs, misconfigurations are more diverse, have fewer specifications, and reside in less structured languages. In the past, dealing with configuration errors relies on manually-encoded rules and checks which are often ad hoc and deficient. My main insight is that configurations are used by the systems; hence, by analyzing the system's code that uses configuration values, one can *automatically* and *systematically* obtain a lot of system-level knowledge of configurations, such as systems constraints of configuration settings and manifestation patterns of configuration errors. Such knowledge is extremely valuable—it enables evaluating configuration design and handling [5], exposing misconfiguration vulnerabilities inside the systems [5], and generating configuration checking code for the systems [6].

Moreover, many misconfigurations stem from human errors. Thus, better configuration *design* is critical to reducing misconfigurations and enabling efficient fixes. In my point of view, configurations are essentially the interface for controlling and customizing systems; however, they are rarely designed like an interface. To make configuration user-friendly and less error-prone, I adopt a user-centric design philosophy and sift through real-world issues to understand sysadmins' difficulties and mistakes in practice [7–9]. This understanding helps identify existing design flaws and reveals the design aspects that need improvement.

Impact. My research has led to more than a hundred code improvements towards better configuration design and implementation—benefiting thousands of configuration parameters—in both commercial and open-source systems, including NetApp storage products, Apache web server, Hadoop, HBase, Squid, MySQL, PostgreSQL, VSFTP, Alluxio, Accumulo, and Redis. Specifically, Spex [5] has motivated Squid web proxy project to improve its configuration library. My configuration usage study [7] was widely quoted in NetApp, and used to make decisions on configuration design for reducing customers' misconfigurations. PCheck [6] received Jay Lepreau Best Paper Award at OSDI 2016.

To make impact, my research addresses important problems, builds practical solutions, and applies the solutions to real-world systems. To validate the importance and relevance of research problems, I constantly interact with developers from both commercial companies and open-source projects, while processing their feedback and opinions. For example, I collaborate closely with NetApp, one major storage vendor for datacenters; I also report issues and propose solutions to open-source projects. Moreover, I make sure that the solutions are practical: all my proposed solutions work with mature, widely-deployed real-world systems and promptly identify problems in these systems.

1 Current Research

My thesis focuses on enabling the design and implementation of *reliable* and *secure* cloud and datacenter systems that confront configuration errors. I first built Spex [5] to automatically infer systems requirements of configurations by analyzing how configurations are used in systems software; I then leveraged the inferred knowledge to harden systems by exposing vulnerabilities that lead to disastrous behavior upon misconfigurations (such as crashes, hangs, and silent failures). From my experience with Spex, I observed that errors in configurations related to fault tolerance and error handling are particularly hard to expose—they only manifest under critical circumstances, leading to catastrophic failure damage. To enable early detection of such errors, I built PCheck [6] to automatically generate checking code which emulates how the system will use configuration values in its late execution and captures anomalies as the evidence of errors. Furthermore, through characterizing and understanding real-world configuration practices, I questioned existing configuration design and identified design flaws that hinder correct configurations [7]. My recent work looks into security misconfigurations [8]. I will now summarize each of these projects.

Hardening systems against configuration errors with Spex. My thesis research started with the observation that many mature and widely-deployed cloud and datacenter systems do not anticipate misconfigurations well and thus are vulnerable to configuration errors. Many configuration errors lead to disastrous behavior such as crashes, hangs, and silent failures, leaving sysadmins clueless but to report the problems to support engineers. Often, support engineers are also misled by the perplexing behavior, causing unnecessarily long repair time upon service disruption and downtime.

I believe the fundamental cause of systems' being vulnerable to configuration errors lies in developers' common attitude towards misconfigurations—unlike software bugs, many developers take a laid-back role in handling misconfigurations but unconsciously assume correct configuration settings. However, sysadmins make mistakes, especially if the configuration requirements are confusing or too strict. With disastrous behavior, it is hard for sysadmins to identify the root causes since they did not write the code and cannot debug the code. On the other hand, if the system is hardened to react gracefully to configuration errors (e.g., printing error messages to pinpoint the erroneous configuration settings), sysadmins can directly fix the errors, hence significantly reduce the diagnosis and repair time.

To urge developers to harden systems against configuration errors, I built Spex to automatically infer configuration requirements (termed constraints) from the source code and then use the inferred constraints to (1) expose misconfiguration vulnerabilities (bad system reactions such as crashes, hangs, and silent failures) and (2) detect certain types of error-prone configuration handling. Spex's insight is that many configuration constraints are reflected in the system's source code, and can be automatically inferred via static code analysis, leveraging the properties of operations and system/library APIs that use configuration values. Spex tracks the data-flow of configuration values in the source code, and looks for patterns of a variety of configuration constraints, including data types, data ranges, control dependencies, and value relationships. With the inferred constraints, I further built a testing tool based on configuration-error injection to expose misconfiguration vulnerabilities—it intentionally violates the inferred constraints to generate errors, and tests how the system reacts (bad reactions are recorded and reported to developers). Moreover, the constraints can be analyzed to detect error-prone configuration handling: namely inconsistency, silent overruling, and unsafe behavior.

Spex has uncovered 743 misconfiguration vulnerabilities of various types and 112 error-prone configuration handling cases in both commercial and open-source systems (more than half have been fixed).

Early detection of configuration errors with PCheck. From my experience with Spex, I observed that errors in configurations related to fault tolerance and error handling are particularly hard to expose. Since these configurations are not needed for initialization or normal operations, many systems do not check their settings early but directly use the values under critical circumstance (e.g., when encountering faults/errors). Thus, the errors become latent until their manifestations cause catastrophes. For such latent errors, early detection is the key to reducing their failure damage. Unfortunately, after studying the configuration checking practices of six mature, widely-deployed cloud and datacenter systems, I found that many critically important configurations (e.g., those used for fault tolerance) do not have any initial checking code to validate the correctness of their settings, and thus are subject to latent errors.

To enable early detection of configuration errors, I built PCheck [6] to automatically generate the configuration checking code and invoke the checking at the system's initialization phase. PCheck exploits the fact that except for explicit checking code, the actual code that uses configuration values (which already exists in source code) can serve as an *implicit* form of checking, e.g., `open`-ing a file path specified by a configuration value implies a capability check. Such usage-implied checking is often more complete and accurate than the explicit checks written by developers, because it precisely captures how the configuration values should be used in actual program execution. Sadly, usage-implied checking is rarely leveraged to detect configuration errors because the usage code often comes too late,

especially if the code is for features like fault tolerance. PCheck generates checking code by emulating the late execution that will use the configuration values; meanwhile capturing any anomalies exposed during the emulated execution as the evidence of configuration errors—same anomalies would occur in real execution, if the errors are not fixed.

The key challenges PCheck faces is to make the checking code *effective* and *safe*. PCheck statically extracts instructions that transform, propagate, and use configuration values from the system program. To execute the extracted instructions, PCheck makes the best effort to determine the values of dependent variables and produce self-contained execution context. To ensure that the checking code is safe to invoke, PCheck sandboxes the emulated execution by instruction rewriting to prevent any side effects on the running system or its environment. Furthermore, PCheck inserts instructions to capture the anomalies that may occur during the emulated execution, based on which it reports errors. PCheck can detect over 75% of real-world latent configuration errors at system initialization, and is generally applicable to cloud and datacenter systems because it does not rely on any predefined rules or external datasets.

Designing configurations to be user-friendly and less error-prone. While defending systems against configuration errors, I realized that many misconfigurations were human errors introduced by sysadmins. This motivated me to seek design principles and disciplines to make configuration less error-prone and more user-friendly. Certainly, such design should follow a user-centric design philosophy.

I believe that one fundamental reason for today's prevalent misconfigurations is the tremendous but still-increasing complexity of configuration, reflected by hundreds or even thousands of configuration parameters ("knobs") exposed by cloud and datacenter systems. Many knobs have various constraints, consistency requirements, and dependencies with other knobs. Such complexity makes it daunting and error-prone to configure cloud and datacenter systems. With the observation that these systems keep adding new knobs without disciplines, I raise the concern that we will be facing more and more knobs and gradually lose the ability to manage the complexity.

I questioned whether the complexity was necessary by studying real-world configuration usage characteristics [7, 9]: Do sysadmins really need so many knobs? Can they manage the complexity? What are their difficulties and mistakes? The answers to these questions conclude that configurations are over-designed—only a small percentage of knobs are set by the majority of sysadmins in the field, while the majority of knobs are seldom touched. Many knobs are neither necessary nor worthwhile—they make configuration more complex but produce little benefit. I observed that with too many knobs sysadmins often have difficulties in finding the right one to achieve intended system behavior or performance goals. Moreover, the excessive number of knobs prevents sysadmins from understanding the configurations thoroughly and examining the settings carefully. Many sysadmins choose to stay with the default values; however, such practices constantly result in misconfigurations that violate the constraints of runtime environment [10].

Based on the understanding learned from the real world, I proposed a few concrete, practical design guidelines which could significantly reduce the configuration space and thus simplify configuration design, with little impact on the desired flexibility. To help navigate existing vast configuration space, I built Cox, a tool that help sysadmins find the right knobs by expressing their intent, based on natural language processing techniques.

Understanding security misconfigurations. My recent work looks into security misconfigurations, one major cause of real-world security failures such as data breaches and server compromises. Even with correctly-coded systems, misconfigurations of security features such as ACLs and firewall can open the door for illegal and malicious access, imposing severe security risks. Unfortunately, security misconfigurations cannot be addressed by tools like Spex and PCheck, as they are not manifested via crashes or exceptions. In reality, they often go undetected until there is a crisis.

To tackle security misconfigurations, I started with understanding real-world security configuration practices in the context of access control, with a focus on how and why misconfigurations were introduced in the field [8]. The study reveals that many real-world security misconfigurations root in sysadmins' difficulties in granting the exact permissions needed for specific functions. With the pressure of resolving access-denied issues that block desired functions, sysadmins tend to grant too much access as workarounds. In the course of examining real-world issues, I believe that the lack of precise and adequate system feedback is the fundamental reason that induces misconfigurations. Therefore, I systematically evaluate the access-control related systems messages in six mature and widely-deployed systems software projects, and find that all of them constantly miss opportunities to provide actionable feedback, creating unnecessary obstacles to secure configurations. With this experience, I have planned a series of future research projects to address the pressing threats of security misconfigurations.

Other research. I am broadly interested in computer systems, and my research is not entirely limited to addressing configuration errors. During my Ph.D., I have also worked on protecting mobile systems from buggy and malicious apps [11–13], cloud storage systems [14], and measurement of Internet services [15].

2 Future Research

I am excited to continue my work in *reliability* and *security* of computer systems, in particular, the emerging cloud and datacenter systems and mobile and IoT systems that have raised many fundamental challenges. I look forward to taking on these challenges, building practical solutions, and making broad impact.

Security configuration and management. My study [8] highlights that security misconfigurations are prevalent and posing significant threats to the security of today's systems, even if the systems are correctly coded. Compared with the extensive work that has been done on detecting vulnerabilities and verifying implementation correctness, existing support for security configuration and management is rudimentary and creates the following barriers to secure configuration settings: (1) imprecise or inadequate systems feedback for permission issues; (2) opaque impact of security configurations—sysadmins are uninformed about potential security risks when changing configurations; and (3) lack of available tools to help sysadmins manage system-level security configurations and resolve security issues.

My vision is that systems support can significantly reduce security misconfigurations—although security configurations are difficult and error-prone to human administrators, they can be and should be automated. I intend to build systems support to facilitate security configuration and management in the following aspects. First, I am interested in enabling the systems to provide precise and adequate feedback (e.g., through log messages) for permission/access-control issues. In this way, sysadmins can nail down the missing permissions and fix problems with the least privileges. Second, I plan to build tool support for measuring security impact of new configurations introduced in the field to forewarn sysadmins about potential security risks. Third, as most security misconfigurations are introduced via configuration changes, I would like to explore approaches that automatically generate configuration "patches," which satisfy the desired functional goals while still maintaining the same level of protection against malice.

Understanding the impact of changes. Large-scale cloud and datacenter systems are never static artifacts but embrace frequent changes of code and configurations. One major difficulty in operating these systems is understanding and anticipating the impact of changes. As cloud and datacenters typically consist of large numbers of heterogeneous components across stacks and nodes, the impact of code and configuration changes often manifests through indirect component interactions and thus cannot be determined by per-component testing. Without the understanding of such impact, changes of code and configurations often lead to unexpected and intricate failures. For example, Welsh described a failure case at Google [16] in which a configuration change in one system component caused resource exhaustion in another system component; due to the scale and complexity of the systems, the engineers spent "many, many hours," trying different things, without success. I intend to help operators and administrators understand and determine the potential impact before rolling out the changes to production. This requires addressing the fundamental challenges in understanding and modeling cross-component, cross-stack interactions in cloud and datacenter systems.

Adapting systems to emerging platforms. Reliability and security are not unique to cloud and datacenter systems, but are desired by the emerging mobile and IoT systems as well. My research on mobile systems [11–13] reveal that these systems have been suffering from severe reliability and security issues in reality, such as abnormal battery drain, cellular overuses, privacy leaks, and storage exhaustion. The key challenges stem from the highly dynamic mobile environment and diverse user patterns. For example, mobile devices are connected to dynamic networks with frequent connectivity disruptions, network switches, and quality changes. However, most existing network configurations (e.g., timeout and retry count) rely on static values which make dynamics fundamentally difficult to accommodate. Also, configurations by definition should be customized for the patterns and preferences of different users, but in practice, they fall short. I am prepared to build systems support to enable reliable and secure mobile and IoT systems that can automatically sense and adapt themselves to the dynamic environment and customize themselves for every individual.

Reliability and security beyond correctness. Building reliable and secure systems is much more than ensuring the correctness of system implementation; it requires understanding the entire ecosystem including operational practices, execution environment, and human operators and users. For example, even correctly-coded systems can be misconfigured and become fragile and vulnerable; without careful design, fail-safety as a desired security principle can cause unnecessary unavailability in cloud and datacenter systems; a mobile OS with correct functions on its own can be shut down by buggy user-level apps that eat up all the battery. As today's systems have been moving into open platforms with shared resources and highly-dependent services, it is critically important to build systems that are aware of errors and issues raised beyond every single program. I aspire to step towards the reliability and security beyond correctness.

References

[1] Summary of the Amazon EC2 and Amazon RDS Service Disruption in the US East Region. https://aws.amazon.com/message/65648/, 2011.

[2] Details of the December 28th, 2012 Windows Azure Storage Disruption in US South. https://azure.microsoft.com/en-us/blog/details-of-the-december-28th-2012-windows-azure-storage-disruption-in-us-south/, 2012.

[3] Poor Documentation Snags Google. http://www.availabilitydigest.com/public_articles/0504/google_power_out.pdf, 2010.

[4] **Tianyin Xu** and Yuanyuan Zhou. Systems Approaches to Tackling Configuration Errors: A Survey. *ACM Computing Surveys (CSUR)*, 47(4), Jul. 2015.

[5] **Tianyin Xu**, Jiaqi Zhang, Peng Huang, Jing Zheng, Tianwei Sheng, Ding Yuan, Yuanyuan Zhou, and Shankar Pasupathy. Do Not Blame Users for Misconfigurations. In *Proceedings of the 24th Symposium on Operating System Principles (SOSP'13)*, Farmington, PA, Nov. 2013.

[6] **Tianyin Xu**, Xinxin Jin, Peng Huang, Yuanyuan Zhou, Shan Lu, Long Jin, and Shankar Pasupathy. Early Detection of Configuration Errors to Reduce Failure Damage. In *Proceedings of the 12th USENIX Symposium on Operating Systems Design and Implementation (OSDI'16)*, Savannah, GA, Nov. 2016.

[7] **Tianyin Xu**, Long Jin, Xuepeng Fan, Yuanyuan Zhou, Shankar Pasupathy, and Rukma Talwadker. Hey, You Have Given Me Too Many Knobs! Understanding and Dealing with Over-Designed Configuration in System Software. In *Proceedings of the 10th Joint Meeting of the European Software Engineering Conference and the ACM SIGSOFT Symposium on the Foundations of Software Engineering (ESEC/FSE'15)*, Bergamo, Italy, Aug. 2015.

[8] **Tianyin Xu**, Han Min Naing, Le Lu, and Yuanyuan Zhou. How Do System Administrators Resolve Access-Denied Issues in the Real World? In *Proceedings of the 35th Annual CHI Conference on Human Factors in Computing Systems (CHI'17)*, Denver, CO, May 2017. (to appear).

[9] **Tianyin Xu**, Vineet Pandey, and Scott Klemmer. An HCI View of Configuration Problems. *CoRR*, abs/1601.01747, Jan. 2016.

[10] Jiaqi Zhang, Lakshmi Renganarayana, Xiaolan Zhang, Niyu Ge, Vasanth Bala, **Tianyin Xu**, and Yuanyuan Zhou. EnCore: Exploiting System Environment and Correlation Information for Misconfiguration Detection. In *Proceedings of the 19th International Conference on Architecture Support for Programming Languages and Operating Systems (ASPLOS'14)*, Salt Lake City, UT, Mar. 2014.

[11] Peng Huang, **Tianyin Xu**, Xinxin Jin, and Yuanyuan Zhou. DefDroid: Towards a More Defensive Mobile OS Against Disruptive App Behavior. In *Proceedings of the 14th International Conference on Mobile Systems, Applications, and Services (MobiSys'16)*, Singapore, Jun. 2016.

[12] Zhenhua Li, Weiwei Wang, **Tianyin Xu**, Xin Zhong, Xiang-Yang Li, Yunhao Liu, Christo Wilson, and Ben Y. Zhao. Exploring Cross-Application Cellular Traffic Optimization with Baidu TrafficGuard. In *Proceedings of the 13th USENIX Symposium on Networked Systems Design and Implementation (NSDI'16)*, Santa Clara, CA, Mar. 2016.

[13] Xinxin Jin, Peng Huang, **Tianyin Xu**, and Yuanyuan Zhou. NChecker: Saving Mobile App Developers from Network Disruptions. In *Proceedings of the 11th ACM European Conference on Computer Systems (EuroSys'16)*, London, UK, Apr. 2016.

[14] Zhenhua Li, Cheng Jin, **Tianyin Xu**, Christo Wilson, Yao Liu, Linsong Cheng, Yunhao Liu, Yafei Dai, and Zhi-Li Zhang. Towards Network-level Efficiency for Cloud Storage Services. In *Proceedings of the 14th ACM Internet Measurement Conference (IMC'14)*, Vancouver, Canada, Nov. 2014.

[15] Zhenhua Li, Christo Wilson, **Tianyin Xu**, Yao Liu, Zhen Lu, and Yinlong Wang. Offline Downloading in China: A Comparative Study. In *Proceedings of the 15th ACM Internet Measurement Conference (IMC'15)*, Tokyo, Japan, Oct. 2015.

[16] Matt Welsh. What I wish systems researchers would work on. http://matt-welsh.blogspot.com/2013/05/what-i-wish-systems-researchers-would.html, 2013.