RESEARCH STATEMENT

Xiao Yu (xyu10@ncsu.edu)

My research interests span the areas of software systems and software engineering. The main theme of my research is understanding, detecting, and diagnosing reliability and performance problems in real-world software systems. My goal has been to make contributions to both software systems and software engineering research by analyzing and abstracting runtime behaviors in complex systems to support advanced program analysis and comprehension. My research is particularly interdisciplinary, often borrowing and adapting techniques from multiple fields, such as program analysis and data mining.

Background and Current Work

Software systems have never been as complex as they are today. Such complexity is in part reflected by having components and layers integrated and stacked together. It is essential for them to interact with each other reliably to make the whole system functional. However, in real-world software systems, component interactions are prone to failures and performance bottlenecks due to unforeseen or unhandled execution characteristics. These characteristics include, but are not limited to, behaviors of third-party components from the software ecosystem (e.g., kernel drivers from different vendors), unstable production environments (e.g., a slow network), and other unexpected system behaviors (e.g., caused by bugs or poor design). It is challenging for developers to identify or predict these characteristics, as some of them may not appear in the development stage. And it is even more challenging to understand and diagnose the caused problems, as performance and privacy concerns in production systems only grant developers limited access, rendering common debugging techniques unusable.

To help developers understand, detect, and diagnose production problems, I attempt to explore different ways by leveraging data that are accessible to developers under limited access, such as logs and traces. These data essentially provide a window to observe system behaviors in the wild, and are widely available in current systems. However, these data are also massive, noisy, and unstructured, thereby posing difficulties for developers to effectively discern information related to specific diagnostic purposes. Therefore, I design approaches to model and abstract essential and useful information from these data, especially data and control flow information in component interactions.

My research includes findings and techniques for unique problems in different software systems: operating systems with performance-affecting third-party components, distributed cloud infrastructures with the need of monitoring production failures and performance degradation, web-based applications with the need of analyzing uncoordinated handling of related requests, and database applications with inefficient database accesses. These systems and applications represent different aspects of the current software ecosystem, in which applications with both web and database characteristics run on top of operating systems and cloud infrastructures.

Analyzing Execution Traces for Bottlenecks in Kernel Drivers

An operating system can be regarded as a software ecosystem, in which multiple components interact with each other. The operating system performance can therefore be affected by problematic components and their interactions with others. And it is difficult to diagnose performance bottlenecks involving component interactions. Such difficulties are mainly reflected by subtle triggering conditions and root causes hidden deeply from places where bottlenecks manifest.

Addressing bottlenecks related to component interactions depends on two key questions: (1) what problematic components and their runtime behaviors are in the real world; and (2) to what extent and how they would affect the overall system performance. In practice, execution traces from real-world systems provide rich information to answer these questions, because they capture usage scenarios with diverse components and interactions that are not seen during in-house testing. However, these traces contain millions of discrete tracing events, rendering inspection without automated techniques ineffective.

In collaboration with Microsoft Research, I had a chance to investigate a large number of execution traces collected by the Windows Customer Experience Improvement Program from 339 hours of real-world application and system usage. My findings included that Windows kernel drivers constitute a non-trivial part in the overall system performance through component interactions including function invocations and synchronizations: (1) a driver can invoke other drivers in the same driver stack through the kernel, and (2) a driver can block other drivers by synchronizations. The combination of these interactions can propagate and amplify performance bottlenecks across multiple components and processes. Based on these findings, I designed two general automated analyses to answer the key questions: (1) impact analysis measuring what drivers and to what extent they impact the overall system performance, and (2) causality analysis identifying interaction patterns of high-impact drivers that are likely to cause perceived performance bottlenecks. On the collected traces, the impact analysis revealed that kernel drivers constitute 36.4% on waiting time and 1.6% on running time, and 26.0% of the driver waiting time is due to the amplification of interactions. The causality analysis further discovered some real-world cases of high-impact drivers with their long interaction paths that lead to significant performance bottlenecks [1].

Monitoring Cloud Infrastructures via Interleaved Logs

Cloud infrastructures provide a rich set of management tasks, whose executions involve interactions and collaborations among multiple distributed components. Monitoring such task executions is crucial for cloud providers to promptly identify and fix problems that compromise cloud availability. When a reliability problem occurs, such as performance degradation, it is particularly useful to know what task execution is affected and what component may be the source of the problem.

Monitoring logs centralized from different components is a common practice. This process is lightweight and non-intrusive, and does not introduce extra complexity or overhead to the system under monitoring. However, even when centralized, logs are still unstructured and highly interleaved due to the distributed and multi-tenant nature of such systems, limiting log monitoring to specific types of log messages, but not associating them with task executions. So I attempt to address this question: is it possible to monitor how individual tasks are being executed by attributing individual log messages to their corresponding task executions?

In collaboration with NEC Laboratories America, I analyzed logs generated from OpenStack, a popular open-source cloud-infrastructure system. Our findings presented both opportunities and challenges for log monitoring: (1) log timestamps can determine the order of distributed components executing the same task, but asynchronous executions also introduce non-deterministic message orders; (2) some resource identifiers shared by log messages can be used to attribute messages to executions, but no single identifier is shared by all messages in the same execution. Based on these observations, I proposed a log-based monitoring approach to monitor workflows of task executions. This approach features: (1) a modeling technique, which abstracts log messages and their possible orders into an automaton model for each task, and (2) an efficient checking algorithm, which uses the modeled automata and partially shared identifiers to accurately attribute interleaved log messages to their corresponding task executions, and reports state divergences as potential failures. The experiments on OpenStack showed that the approach can correctly and efficiently associate at least 92.08% of the task executions with their log messages, and can achieve a precision of 83.08% and a recall of 90.00% on the injected problems [2].

Analyzing and Understanding Request-based Characteristics in Web-based applications

Web-based applications serve as the front end of server applications, such as web sites and RESTful APIs. These applications adopt the request-based execution model, which typically consists of a set of request-handler methods invoked upon user requests by an application framework. In this model, highly modular and stateless methods often lead to repetitive and uncoordinated application behaviors across requests. For example, methods handling two related requests may repeatedly create identical data objects without the knowledge of request-input dependencies. To implement and further optimize these methods with efficiency, it is crucial for developers to understand what data dependencies may exist between related request-handler methods.

Conventional interprocedural analysis is ineffective to infer and analyze such data dependencies, because they along with method calling relationships are implicitly established by the complex framework and request sequences. In addition, components involved in propagating data from request input to output are usually written in different languages, e.g., request-handler methods in ordinary object-oriented languages, while view templates rendering request output in markup languages.

To solve these analysis challenges, I proposed a dynamic approach to capture indirect data dependencies between request-handler methods. This approach uses a lightweight tracing technique to capture how object data flow along complex framework behaviors without instrumenting the framework code, and a mining technique to reconstruct data dependencies. I showed that the reconstructed dependencies are over 80% accurate and useful in understanding the behaviors of request-handler methods in the inter-request scope, such as tuning object caching policies. I envision this approach as a foundation for future full-fledged analysis tools [4].

Understanding Characteristics of Performance Bugs in Database Accesses

Another prevalent characteristic of web-based applications is the use of databases, leading to heavy database accesses. A database access usually involves many components, making it prone to performance bugs: (1) the application issuing queries, (2) queries themselves consisting of a set of database operations, (3) the database engine executing queries, and (4) the database schema and stored data. An exposed performance bug may be attributed to any of these components. To understand such bugs for future analysis and repairing techniques, I analyzed over 130 bug reports from real-world open-source projects. The result reveals root causes, triggering conditions, and fixes of the studied bugs. It further suggests that these bugs are closely related to developers' imprecise assumptions on query executions, workload, and stored data. For example, the design of application logic or related queries may be based on an imprecise assumption on the cardinality of queried data, causing either costly iterations of data accesses in the application, or expensive query executions in the database engine. Based on my findings, I believe that future research can advance in two directions. On the application side, we need techniques to assist developers to understand potential workload and data characteristics for efficient database accesses. On the database side, we need practical query-optimization techniques that can adapt to shifts in real-world workload and data [3].

Research Agenda

My current research reveals a common problem pattern in complex software systems. On the one hand, layered and modular systems make software development easier and less prone to correctness bugs by hiding and decoupling internal complexity. On the other hand, they can introduce reliability, performance, and diagnosability challenges. Built upon the insights from my current research, I plan to further address these challenges and the caused problems with practical and elegant solutions in my future research.

Short-Term Goal: Addressing Cross-Component Reliability and Performance Problems

In the short term, I am interested in developing techniques that leverage the insights from my current research to help developers diagnose and resolve reliability problems related to component interactions in existing systems. In particular, I propose the following three tasks in the direction of this goal.

Modeling Cross-Component Control and Data Flows. Modeling control and data flows across components and layers in complex systems is the first step towards my short-term goal. My existing work has shown that such modeling is particularly important and fundamental to analyze and understand system behaviors and problems: (1) synchronization-related behaviors across kernel drivers and operating system performance [1], (2) task workflows across distributed components and cloud-infrastructure monitoring [2], and (3) data dependencies between highly modular request-handler methods and inter-reqeust analysis of web-based applications [4]. I plan to further investigate this direction by including and combining more components and layers into the modeling, such as data dependencies between application components and database, and end-to-end control and data flows across layers from high-level applications to low-level operating systems.

Bridging External Characteristics and System Behaviors. In production systems, we are usually able to find out what happened internally in failures or performance degradation through log and trace analysis. However, it is not straightforward to figure out what and how external characteristics (e.g., environments, input, and workload) caused the observed problems, as connections between these characteristics and problematic system behaviors are not clear. For example, we can identify synchronization-related performance bottlenecks in kernel drivers, and failures in cloud infrastructures, but we do not have the input information to reproduce them for further debugging and analysis. Consequently, debugging and other recovery actions become costly, and rely more on the expertise of developers. This challenge can be addressed by analyzing and providing information on how systems would behave under different environments, input, and workload. I plan to explore and develop the required techniques based on the modeled control and data flows. In my current research on database-related performance bugs, a concrete application of such intended techniques can be used to infer how data characteristics would affect the performance of database accesses.

Improving Diagnosability of Cross-Component Reliability and Performance Problems. The aforementioned two tasks will empower the advancement of diagnosis techniques for cross-component reliability and performance problems, and I plan to develop these techniques. My existing work has been able to detect failures and performance problems in different systems, and provide useful information for diagnosis. The intended diagnosis techniques will further advance my existing work. For instance, I intend to investigate techniques to pinpoint root causes and provide information for reproducing when failures are identified in logs and traces. For performance problems in web-based applications, I intend to investigate techniques to diagnose and optimize inefficient data accesses across request-handler methods and application-database layers. These techniques will leverage inter-request data dependencies and other application-specific data characteristics, and benefit from techniques in the aforementioned two tasks.

Long-Term Goal: Building Self-Recoverable and Self-Optimized Systems

In the long term, I plan to bring in fundamental architectural changes for self-recoverable and self-optimized systems, which can self-optimize based on different system characteristics, and self-recover from operation failures. I believe that such autonomous systems will be practical and prevalent in future, because software development has always been made easier and more swift by abstracting and hiding low-level details, and this trend will ultimately expand to debugging, tuning, and operation.

My current research has started building the necessary blocks for this long-term goal. For instance, although my log and trace analysis techniques for operating systems and cloud infrastructures are developer- and administrator-oriented, it is possible and desirable that machines can also interpret analysis results, and take actions accordingly. My work on applications with the request-based execution model essentially reveals cross-component and cross-request data dependencies, which can potentially be used by machines in determining how components should be organized and collaborate with each other. My long-term goal requires more deep understanding on different systems, and techniques for systems to learn and adjust themselves. I intend to iteratively review and investigate this goal with accumulated knowledge along my research in achieving my short-term goal.

Overall, my future research will include both a practical part for existing systems, and a futurist part targeting at my long-term goal. This combination allows my research to be relevant and impactful to the industry, while have the vision to push forward the boundary of systems research. My collaborations with industrial research laboratories have given me an advantage of seeing and understanding practical problems. With the accumulated understanding, I am confident and excited at paving the future path of my research, and making an impact in the related fields.

References

[1] Xiao Yu, Shi Han, Dongmei Zhang, and Tao Xie. Comprehending Performance from Real-World Execution Traces: A Device-Driver Case. In *Proceedings of the 19th International Conference on Architectural Support for Programming Languages and Operating Systems*, ASPLOS '14, pages 193–206, New York, NY, USA, 2014. ACM.

[2] Xiao Yu, Pallavi Joshi, Jianwu Xu, Guoliang Jin, Hui Zhang, and Guofei Jiang. CloudSeer: Workflow Monitoring of Cloud Infrastructures via Interleaved Logs. In *Proceedings of the 21st International Conference on Architectural Support for Programming Languages and Operating Systems*, ASPLOS '16, pages 489–502, New York, NY, USA, 2016. ACM.

[3] Xiao Yu, Wei Yang, Mengqi Gu, Guoliang Jin, Tao Xie, and Xintao Wu. Characterizing Performance Bugs in Database Applications. Submission Under Review, 2017.

[4] Xiao Yu and Guoliang Jin. Dataflow Tunneling: Mining Inter-request Data Dependencies for Request-based Applications. In *Proceedings of the 40th International Conference on Software Engineering*, ICSE '18, 2018. To appear.