Research Statement: Karthik Pattabiraman, University of British Columbia (UBC)

Dependable computer systems are those that continue to operate correctly despite the presence of errors and security attacks. Traditionally, dependable systems have been confined to domains such as aerospace, finance and health-care, where cost was not a significant factor to achieve high dependability. However, in recent times, dependability has become a pervasive concern for commodity computer systems, where cost, performance and energy-efficiency are the key constraints that drive technological innovation. For instance, data-centers, desktop computers and mobile phones all have to be designed and built around these constraints. The primary goals of my research program are to design, build and validate dependable commodity systems that are fast, efficient and inexpensive. I call this paradigm "good enough dependability".

My PhD thesis at the University of Illinois [Thesis] laid the foundations of "good enough dependability", by developing application-level solutions to hardware faults (e.g., soft errors). In contrast to traditional approaches that duplicate the entire program to protect it from soft errors, in my thesis, I proposed a paradigm shift by developing automated methods to identify important data in the application and protect it using static and dynamic analysis based methods. I showed that it is possible to achieve nearly the same coverage as full duplication like approaches, for only a fraction of the cost in terms of performance overheads. This approach has been prototyped in the Trusted Illiac platform developed at Illinois. I was awarded the William Carter award by the IEEE Technical Committee on Fault Tolerant Computing (TC-FTC) in 2008 based on my dissertation research.

After my PhD, I joined Microsoft Research (MSR) as a post-doctoral researcher, where I led a new direction on saving energy in mobile phones by partitioning application data into critical and non-critical, and then allocating the non-critical data on less reliable, but highly energy efficient, DRAM memory. This approach, which we called Flikker [ASPLOS'11], was the first to explicitly leverage the differential reliability of application data to save energy. Since our work, other groups have built on this idea, for example, at the University of Washington, and at MIT. At MSR, I also led a new direction on characterizing the reliability of JavaScript-based web applications [ISSRE'10], and on partial type-safety for security [CSF'11]. The latter was invited as one of the four best papers presented at the CSF'11 conference (of more than 80 submitted) to the Journal of Computer Security [JCS].

I then joined the University of British Columbia (UBC) as a tenure-track assistant professor in the Electrical and Computer Engineering department. At UBC, I lead a research group consisting of about 10 graduate students (7 PhD, 3 Masters). In addition, I have graduated a total of 8 graduate students (1 PhD, 7 Masters). My students are all funded by my research grants from Canadian government agencies (e.g., NSERC, CFI, MITACS), and from industry. I have collaborated and secured funding from leading industry labs such as Microsoft, Intel, IBM, Cisco, AMD and Lockheed Martin. Some of my research has been successfully transitioned to industry, and my group has developed and released multiple open-source tools that are widely used by other researchers.

At UBC, along with my students and colleagues, I have explored three directions in the "good enough dependability" paradigm. The central tenet that runs through all three directions is to systematically protect the highest value targets in an application from errors and attacks subject to a given cost, energy or performance overhead. The first direction is to build error resilient applications by identifying and selectively protecting the important data in the application from hardware faults. The second direction is to identify the most severe reliability issues in modern JavaScript-based web applications, and to target these issues for mitigation. The third direction is to protect smart embedded devices from security attacks by monitoring the most security-critical system calls in the application from intrusions. I first describe these three directions below, and then outline my plans for future work.

1. Error resilient Applications

Hardware errors have become more prevalent in recent times due to the effects of technology scaling. It is projected that hardware error rates will continue to increase due to shrinking feature sizes, increasing manufacturing violations and temperature hotspots. The traditional way to handle hardware errors is through redundant hardware, i.e., by using dual-modular or triple-modular redundancy. However, these methods incur prohibitive energy overheads, which make them impractical, as energy is becoming a first class design constraint [Borkar'11]. An alternate approach is to expose errors up the system stack to the application software, and mitigate the errors at the software level. The main advantage of this approach is that very few errors propagate up the stack to the application (less than 10% [Sagesse'05]), and hence it is much more efficient to tolerate the errors at the application level. The challenge is that the software needs to be made error resilient, which is difficult for programmers, as they lack appropriate techniques for doing so.

My research develops novel approaches to help programmers make their software applications resilient to hardware faults. I have explored three lines of research in this area, with my graduate students (3 PhD, 3 Masters), colleagues at UBC, and researchers in industry.

The first line of research identifies important data in the application that should be protected from transient hardware faults. When an application experiences a fault, it may or may not fail, depending on where the fault occurs and when. Failures of applications can be classified into crashes, hangs and Silent Data Corruptions (SDCs). Of these outcomes, SDCs or incorrect outcomes are often the most serious, as there is no external indication that an error has occurred. The main insight in our work is that SDCs predominantly occur due to errors affecting just a few data variables of the program, and that such variables can be automatically identified through static analysis techniques. Our approach identifies program features that correlate with the SDC-proneness of program variables. It then uses these features to find the most SDC-prone variables, and selectively protects them by duplicating the backward slice of these variables in the program. We find that our "good enough" approach can provide high coverage for SDC-causing errors, at only a small fraction of the cost of full duplication [DSN 2012, DSN 2013, CASES 2014]. Some of this work was conducted in collaboration with IBM T.J. Watson Research.

The second line of research is to mitigate intermittent hardware faults using an integrated hardware-software method. Intermittent faults are those caused by manufacturing variations, wear-out and temperature hotpots, and have gained a lot of prominence, with a recent study from Microsoft showing that they account for 40% of hardware faults [Nightingale11]. Unlike transient faults they repeatedly occur at the same location, although in non-deterministic ways, and hence need diagnosis. However, performing diagnosis is challenging due to their non-determinism. Our approach uses a lightweight hardware component to record the program's execution in the processor's pipeline stages and execution units. When the program fails (crashes) due to an intermittent fault, the information gathered by this component is exposed to the software level, which uses the information to identify the probable root cause of the failure. We find that our "good enough" approach can achieve high accuracy with low power and performance overheads, compared to traditional approaches [DSN 2014a, TR].

The third line of research is to build robust fault-injection tools and techniques for empirical evaluation of error resilient applications. This is important because "good enough" techniques by definition do not provide 100% coverage, and hence need to be rigorously quantified to provide a principled tradeoff between coverage and overheads. One of the fault-injection tools we have built, LLFI (integrated with the LLVM compiler), allows fine-grained customizability of where and when to inject faults in the program, and allows programmers to trace the propagation of faults after injection. We have shown that LLFI is accurate for injecting SDC-causing faults in the program [DSN14b], although it operates at a much higher level of abstraction than other techniques. Many groups in academia are actively using LLFI. Further, Cisco Systems has been working with us to internally adopt LLFI.

Another fault-injection tool we have developed called GPU-Qin, allows one to evaluate the error resilience of general-purpose applications executing on Graphic Processing Units (GPUs) [ISPASS2014, DATE2014]. GPU-Qin was developed jointly with AMD Research, and is being used by groups at national labs, academia and industry. GPU-Qin is the first fault injector for general-purpose programs executing on GPUs, which requires neither expensive simulations, nor modifications to the program's source code, and is hence both efficient and portable.

The above projects have received funding from the Canadian funding agency (e.g., NSERC), from Lockheed Martin Company, Cisco Systems and AMD Research. Our work in this area has appeared in top tier conferences such as DSN, and has been widely cited by other researchers working in this area.

Ably supported by a group of dedicated graduate students, research colleagues and industry partners, my work on building error resilient techniques has made fundamental advances in static program analysis, machine learning, and empirical reliability evaluation. Our work was the first to show that using the "good enough" dependability approach, one can build error resilient applications that tolerate most hardware errors, for only a fraction of the cost of traditional techniques.

2. Web Application Reliability

Modern web applications have become pervasive and are already replacing traditional desktop applications. This is because they are extremely interactive with rich media content, and can emulate the look and feel of many native applications, while avoiding the hassles of installation. This interactivity is achieved with the use of client-side

code written in the JavaScript programming language, which is interpreted and executed within the web browser. JavaScript is a dynamically typed, very permissive language, which places few restrictions on what programs can do. As a result, code written in JavaScript is believed to be highly error prone [Richards11].

My research focuses on improving the reliability of modern web applications written using JavaScript. Along with my students and colleagues at UBC and Microsoft Research, I studied the reliability of JavaScript code in the Alexa top 100 most popular websites. We found that even mature, popular websites experience multiple JavaScript errors during their execution, and these errors are hidden behind the JavaScript console [ISSRE'11]. However, the web application continues to execute in spite of these errors, suggesting that not all errors are severe.

To understand which JavaScript errors are the most severe, we empirically studied the bug reports of 12 open source JavaScript applications. We found that most of the severe bugs fall into the category of DOM-JavaScript interactions, or interactions with the Document Object Model (DOM) of the web application [ESEM'13]. The DOM is a hierarchical representation of the web page that dynamically evolves as the web application executes. JavaScript applications frequently interact with the DOM of the web application. We found that these interactions are error prone as programmers often do not have a clear mental model to reason about the DOM. Our study thus overturns the widely-held notion that errors in JavaScript-based web applications are due to features of the JavaScript language, and instead finds that they are due to interactions with the environment (i.e., the DOM).

We have since focused our research on understanding, repairing and preventing DOM-related JavaScript faults. To help developers understand DOM-JavaScript interactions and visualize them, we built Clematis, a tool to capture the dynamic dependencies between JavaScript code and the DOM [ISCE14a]. We found that Clematis dramatically reduced the time taken by developers to locate program features and bugs, while increasing their accuracy. Clematis was awarded the ACM distinguished paper award at the IEEE/ACM International Conference on Software Engineering (ICSE), 2014, an honor given to only nine papers out of nearly 500 papers submitted to the conference. A leading software company in Vancouver is currently exploring the adoption of Clematis.

To help developers repair bugs in DOM-JavaScript interactions, we built Vejovis, which uses past patterns of bug fixes and a SAT solver to find the optimal bug fix for web applications [ICSE'14b]. Vejovis was able to generate fixes matching human-written fixes in over 90% of the cases across 12 web applications. Finally, to prevent programmers from making DOM-related errors in their code, we have built Dompletion, an automated code-completion system for DOM-JavaScript interactions [ASE14]. We find that Dompletion is accurate at providing code completion suggestions that are useful to programmers, while taking only 2-3 seconds to do so.

All of the above work has been performed with my graduate students (2 PhD, 2 Masters), and colleagues at UBC. This research was supported with funding from Canadian funding agencies (e.g., NSERC), Microsoft Research, and Intel Research (three years in a row). Our papers in this area [ICST12, ICST13] have been respectively nominated for, and won awards at the IEEE International Conference on Software Testing (ICST), 2012 and 2013. We have also transitioned the results of this research to companies, and have published the work in top-tier software engineering conferences such as ICSE.

In summary, my work on understanding and improving the reliability of web applications has been a bellwether in this area, and has spawned several other projects. The distinguishing feature of this work has been the use of empirical studies to identify the most severe reliability problems, and selectively mitigate them. Such an empirical approach to characterizing reliability issues has allowed us to achieve "good enough" dependability for JavaScript applications at low costs.

3. Smart Embedded Devices Security

Smart devices are becoming ubiquitous; yet their security is a challenging problem. These devices range from small motes to smart appliances in homes. Examples of smart devices are smart electric meters, which are replacing traditional analog meters, to obtain real-time, electricity consumption data. Unfortunately, smart meters are vulnerable to attacks as they are inexpensive devices with low computational capabilities, and hence cannot run full-fledged intrusion detection systems (IDS), which incur significant performance overheads.

Along with my students, I have built a low-overhead IDS for smart meters that considers the functionality of the smart meter to decide what aspects of it should be monitored [HASE'14]. Our approach starts from a high-level specification of the smart meter's functionality, and automatically translates the specifications to a checkable model

for the IDS by statically analyzing the smart meter's software. We show that our method provides protection from the most severe security attacks, while incurring only a fraction of the overheads of full-fledged IDSes. Here again, we employ the "good enough" dependability paradigm by selectively providing protection for the most severe security attacks at low cost. This project is funded by Canadian funding agencies.

4. Future Work

I believe that the need for dependability will extend across a broad range of future systems, each with its unique constraints, and hence we need to provide customized solutions for each. To that end, my future work will encompass a broad range of systems from mobile phones to clouds, and explore systematic approaches to make them more dependable. I plan to explore the following three directions in my future work.

Abstractions and algorithms for error resilience: The main challenge with building error resilient applications is that there is a wide array of options for the programmer to choose from in terms of code structures and algorithms, each of which has different resilience. Unlike performance engineering, where there are clear abstractions for programmers to reason about performance, there are no such abstractions in error resilience. In my future work, I will develop such abstractions using a principled approach with insights gathered from empirical studies to characterize and quantify error resilience of different aspects of applications such as the algorithms employed, and the programming patterns used in these applications. These abstractions will allow programmers to reason about their application's error resilience, and make informed choices about what parts of the program to protect. Our work on error resilience of GPU applications [ISPASS'14] partially examined this phenomenon, but there is a lot more to be done. In the long term, I would like to develop a theory of algorithmic error resilience much like asymptotic analysis for performance, and automatically synthesize error resilient algorithms based on the theory. I plan to explore the use of genetic algorithms and program synthesis techniques toward this endeavor.

Hybrid web applications: Similar to how traditional web applications are replacing desktop applications, hybrid web applications are replacing mobile applications. These are applications that use web technologies such as JavaScript, HTML, CSS, but also make use of platform specific features of mobile devices. While hybrid web applications share many of the challenges of traditional web applications, they also engender a new set of challenges. First, their enhanced access to the device's capabilities means that they are more vulnerable to faults and security attacks. Second, in addition to the DOM, there are many other locations where the state of the application is stored (e.g., mobile device's local storage). As we have seen, managing interactions with the DOM is already challenging for programmers; these additional locations make the problem even more complex. Finally, testing hybrid web applications is more difficult than testing traditional web applications, as they interact extensively with the mobile device. To counter these challenges, I will use symbolic execution techniques to build an abstract model of the mobile device, and use the model to enhance the reliability of the hybrid web application.

Cloud applications' dependability: Cloud systems have become pervasive, and more and more software applications are moving to the cloud. However, due to their large-scale, distributed nature, and the use of commodity components, cloud systems are prone to failures and security attacks. It is therefore vital to ensure that cloud applications can continue to execute reliably and securely despite the presence of errors and attacks. Our preliminary work in this area [ISSRE'14] has found that cloud systems waste a considerable amount of resources due to failures of jobs. Further, we find that jobs manifest failure symptoms even halfway during the executions, pointing to the potential for failure prediction, and that cloud applications exhibit regular resource usage patterns, pointing to the potential for anomaly detection. I will leverage these observations to build error and attack resilient cloud applications that can provide customized levels of reliability and security according to users' requirements.

References [Names of Students I advise(d) are bolded in the list of my papers]

[Thesis] *Automated Derivation of Application-aware Error Detectors,* Karthik Pattabiraman, PhD Dissertation, Dept. of Computer Science, University of Illinois at Urbana-Champaign, May 2009.

[ASPLOS'11] *Flikker: Saving DRAM Refresh-power through Critical Data Partitioning,* ***Song Liu,*** Karthik Pattabiraman, Thomas Moscibroda and Benjamin Zorn, Proceedings of the ACM International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS) , California, 2011. *(Acceptance Rate: 20%).* pages: 213- 224.

[ISSRE'10] *DoDOM: Leveraging DOM Invariants for Robustness Testing of Web 2.0 Applications*, *Karthik Pattabiraman* and Benjamin Zorn, Proceedings of the IEEE International Symposium on Software Reliability Engineering (ISSRE), California. 2010. (Acceptance rate: 32 %),

[CSF'11] *Modular Protections against Non-control Data Attacks, Cole Schlesinger*, Karthik Pattabiraman, Nikhil Swamy, David Walker and Benjamin Zorn, Proceedings of the IEEE Computer Security Foundations (CSF) Symposium, France, 2011 (*Acceptance Rate: 26%)*.

[JCS] *Modular Protections against Non-control Data Attacks,* Cole Schlesinger, Karthik Pattabiraman, Nikhil Swamy, David Walker, and Benjamin Zorn, Journal of Computer Security (JCS), November 2014.

[DSN'12] *BlockWatch: Leveraging Similarity in Parallel Programs for Error Detection,* ***Jiesheng Wei*** and Karthik Pattabiraman, Proceedings of the IEEE/IFIP International Conference on Dependable Systems and Networks (DSN), 2012, Boston, USA (*Acceptance rate: 17%)*

[DSN'13] *Error Detector Placement for Soft Computation,* ***Anna Thomas*** and Karthik Pattabiraman, Proceedings of the IEEE/IFIP International Conference on Dependable Systems and Networks (DSN), 2013, Budapest, Hungary. (Acceptance Rate: 20%),

[CASES'14] *SDCTune: A Model for Predicting the SDC Proneness of an Application for Configurable Protection*, **Qining Lu**, Karthik Pattabiraman, Meeta S. Gupta and Jude A. Rivers, International Conference on Compilers, Architecture and Synthesis for Embedded Systems (CASES), 2014. (Acceptance Rate: 30%)

[DSN'14a] *Integrated Hardware-Software Diagnosis for Intermittent Hardware Faults,* **Majid Dadashi, Layali Rashid,** Karthik Pattabiraman and Sathish Gopalakrishnan, Proceedings of the IEEE/IFIP International Conference on Dependable Systems and Networks (DSN), 2014. (Acceptance Rate: 30%)

[TR] *Characterizing the Impact of Intermittent Hardware Faults on Programs*, **Layali Rashid**, Karthik Pattabiraman and Sathish Gopalakrishnan, To appear in the IEEE Transactions on Reliability (TR).

[DSN'14b] *Quantifying the Accuracy of High-Level Fault Injection Techniques for Hardware Faults,* **Jiesheng Wei, Anna Thomas, Guanpeng Li,** and Karthik Pattabiraman, Proceedings of the IEEE/IFIP International Conference on Dependable Systems and Networks (DSN), 2014. (Acceptance Rate: 30%)

[ISPASS'14] *GPU-Qin: A Methodlogy for Evaluating the Error Resilience of GPGPU Applications,* ***Bo Fang*****,** Karthik Pattabiraman, Matei Ripeanu and Sudhanva Gurumurthi, Proceedings of the IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS), 2014, Monterrey, CA (Acceptance Rate: 30%).

[DATE'14] *GPUS: Combining high-performance with high-reliability,* L. Bautista Gomez, F. Cappello, L. Carro, N. DeBardeleben, **B. Fang**, S. Gurumurthi, K. Pattabiraman, P. Rech, M. Sonza Reorda, Embedded tutorial paper, Proceedings of the International Symposium on Design Automation and Test in Europe (DATE), 2014

[ISSRE'11] *JavaScript Errors in the Wild: An Empirical Study*, ***Frolin Ocariza.,*** Karthik Pattabiraman and Benjamin Zorn, Proceedings of the International Symposium on Software Reliability Engineering (ISSRE), 2011, Hiroshima, Japan (*Acceptance Rate: 25%).*

[ESEM'13] *An Empirical Study of Client-Side JavaScript Bugs*, ***Frolin Ocariza*****, Kartik Bajaj,** Karthik Pattabiraman and Ali Mesbah, Proceedings of the IEEE International Symposium on Empirical Software Engineering and Measurement (ESEM), 2013, March, 2013 (Acceptance rate: 28 %).

[ICSE'14a] *Vejovis: Suggesting Fixes for JavaScript Faults,* **Frolin Ocariza,** Karthik Pattabiraman and Ali Mesbah, Proceedings of the IEEE/ACM International Conference on Software Engineering (ICSE), 2014, Hyderabad, India (Acceptance Rate: 20%).

[ICSE'14b] *Understanding JavaScript Event-Based Interactions,* **Saba Alimadi, Sheldon Sequira,** Ali Mesbah and Karthik Pattabiraman, Proceedings of the IEEE/ACM International Conference on Software Engineering (ICSE), 2014, Hyderabad, India (Acceptance Rate: 20%).

[ASE'14] *DOMpletion: DOM-Aware JavaScript Code Completion*, **Kartik Bajaj,** Karthik Pattabiraman and Ali Mesbah, Proceedings of the ACM International Conference on Automated Software Engineering (ASE), 2014. (Acceptance Rate: 20%)

[ICST'12] *AutoFlox: An Automatic Fault Localizer For JavaScript,* ***Frolin Ocariza***, Karthik Pattabiraman and Ali Mesbah, IEEE International Conference on Software Testing, Verification and Validation (ICST), 2012, Montreal, Canada. (*Acceptance rate: 27%)*.

[ICST'13] *Efficient JavaScript Mutation Testing*, ***Shabnam Mirshokraie****,* Ali Mesbah and Karthik Pattabiraman, Proceedings of the IEEE International Conference on Software Testing, Verification and Validation (ICST), 2013, Luxembourg. (Acceptance Rate: 25 %).

[HASE'14] *Model-based Intrusion Detection for Smart Meters*, **Farid M. Tabrizi** and *Karthik Pattabiraman*, Proceedings of the IEEE International Symposium on High Assurance Systems Engineering (HASE), 2014. Miami, USA (Acceptance rate: 30%)

[ISSRE'14] *Failure Analysis of Jobs in Compute Clouds: A Google Cluster Case Study****,*** **Xin Chen**, Charng-da Lu and Karthik Pattabiraman, To appear in the Proceedings of the 25th IEEE International Symposium on Software Reliability Engineering (ISSRE), 2014. (Acceptance rate: 25%).

[Borkar'11] Shekhar Borkar and Andrew A. Chien. 2011. The future of microprocessors. Commun. ACM 54, 5 (May 2011), 67-77.

[Sagesse'05] Giacinto Paolo Saggese, Nicholas J. Wang, Zbigniew Kalbarczyk, Sanjay J. Patel, Ravishankar K. Iyer: An Experimental Study of Soft Errors in Microprocessors. IEEE Micro 25(6): 30-39 (2005).

[Nightingale'11] Edmund B. Nightingale, John R. Douceur, and Vince Orgovan. 2011. Cycles, cells and platters: an empirical analysisof hardware failures on a million consumer PCs. In *Proceedings of the sixth conference on Computer systems* (EuroSys '11). ACM, New York, NY, USA, 343-356.

[Richards'10] Gregor Richards, Sylvain Lebresne, Brian Burg, and Jan Vitek. 2010. An analysis of the dynamic behavior of JavaScript programs. *SIGPLAN Not.* 45, 6 (June 2010), 1-12.