Good Enough Dependability: A new paradigm for building dependable systems

As computer systems become more and more entrenched in our society, ensuring their dependability is paramount to our overall well being. However, ensuring the dependability of complex, real-world systems is challenging. Dependability is often treated as an "all-or-nothing" proposition, with current techniques seeking to provide complete coverage. This proposition leads to high protection overheads in performance or power consumption. In contrast, my research proposes the idea of "good enough dependability", where dependability is achieved in an incremental and on-demand manner, subject to performance and power overhead bounds. The key innovation is to identify the most important errors or security vulnerabilities in a system, and to selectively target them to achieve dependability at low performance and power overheads.

The foundations of the good-enough dependability approach were laid in my PhD dissertation at the University of Illinois (UIUC), where I explored application-level techniques for targeted protection from soft (hardware) errors. My work showed that by leveraging the properties of the software application, one can achieve nearly the same coverage as all-or-nothing techniques such as full duplication for a small fraction of the overhead [1][2][3]. This approach has led to much follow-up work in the dependability community, such as SWAT [4] (UIUC) and Shoestring [5] (U. Michigan). Based on my thesis work, I was awarded the William Carter award by the IEEE Technical Committee on Fault-tolerant Computing (TC-FTC), and the IFIP Working Group on Dependability (WG 10.4), which is the most prestigious award in the field of dependable computing for PhD students. My research was also integrated into the Trusted Illiac cluster prototype at Illinois, which uses reconfigurable logic for runtime error and attack detection [6].

After my PhD, I spent a year as a post-doctoral researcher at Microsoft Research. While there, I initiated and led the Flikker project, which aims to save energy in mobile computing systems by deliberately lowering DRAM refresh frequency, and partitioning application data into critical and non-critical to tolerate the resulting errors. Flikker [7] was one of the first papers in the field of what is now known as approximate computing, and has spawned off numerous follow-up work such as the EnerJ project (at UW) [8] and Rely (at MIT) [9]. Flikker showed that it was possible to achieve significant energy savings by exposing hardware errors to the application, and selectively tolerating the most important errors by using programmer supplied annotations. Flikker thus established the feasibility and promise of the good-enough dependability approach.

Since joining UBC in 2010 as an assistant professor, I have pushed the boundaries of the good enough dependability paradigm in three different areas: (1) Software approaches for building applications that are resilient to hardware faults, (2) Security of smart, resource-constrained, embedded devices, and (3) Software bugs in JavaScript-enabled web applications and their mitigation. These represent three different areas with widely different requirements and constraints, thus showing the versatility of the good enough dependability paradigm. I have also collaborated closely with and received funding from industry to transition my research into real-world systems. I detail the three directions below, followed by future work.

Direction 1: Software Approaches for Building 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. Along with my students and colleagues at UBC, I have developed novel techniques to tolerate hardware errors in software. The main rationale is that only a small fraction of hardware faults propagate up the system stack and cause the application to fail catastrophically, and hence only these faults need to be tolerated.

Our main observation is that there is a strong correlation between the program-level features of application data (e.g., its backward dependencies), and the propensity of data to result in a Silent Data Corruption (SDC) if corrupted. SDCs represent incorrect application outputs and are often the most insidious failure type – therefore, we focus on SDCs. My research uses static and dynamic analysis to identify SDC-causing application data, and selectively protect the data, without requiring the programmer to provide annotations/directives. We have demonstrated that for a given overhead bound specified by the user, our techniques can achieve significantly higher coverage than duplication-based techniques in a wide variety of settings [10][11][12][13][14]. This work has been published in venues such as the IEEE/IFIP International Conference on Dependable Systems and Networks (DSN), the premier venue for work in the dependability area.

These projects have been carried out in collaboration with companies such as IBM T.J Watson Research Center, AMD Research, and Cisco Systems. We have also developed a number of software artifacts for dependability evaluation (i.e., fault injection) such as LLFI [15] and GPU-Qin [16] that we have distributed as open-source software. These have been widely used by researchers in academia (e.g., MIT, Georgia Tech), industry (e.g., AMD, Cisco, IBM and Nvidia) and national labs (e.g., Los Alamos National Labs, Pacific Northwestern National Labs).

Direction 2: Low-Overhead Security for Smart, Embedded Devices

A multitude of smart devices such as thermostats and smart meters are being connected to external networks via the Internet, a phenomenon known as the Internet of Things (IoT). While this interconnectedness provides enhanced functionality to the end user, it makes the devices much more vulnerable to security attacks. The pervasiveness of smart devices, and their limited memory and computing capacity, make them challenging to protect from attacks. Further, the scale of deployment of these devices makes it necessary to keep protection costs low.

In this line of work, we apply the good enough dependability paradigm to protect the most security-critical data in smart devices, and to detect the most severe security attacks. Our goal is to achieve cost-effective protection without any hardware modifications. To this end, we statically (1) identify the program state that an attacker is likely to target, and selectively protect the state, (2) identify the transitions between different states of the software, and ensure that the runtime execution of the system corresponds to its state transition graph. Both of these steps are carried out using an automated compiler-based analysis, with only high-level intervention from programmers. We have demonstrated these ideas in the context of a smart meter (for power grids), and have shown that the techniques incur low performance overheads, while providing sufficiently high coverage for the most critical security attacks.

I started this line of work when I was at MSR, along with colleagues from Princeton University and MSR. Since joining UBC, I have continued working on this direction with my PhD students. This work has been published at conferences such as Computer Security Foundations (CSF), 2011 [16], and the European Dependable Computing Conference (EDCC), 2015 [18]. The former paper was invited as one of the best papers at the conference to a special issue of a journal of computer security (JCS) [19] (one of five papers), while the latter paper was awarded a distinguished paper award at the conference (one of three papers). We are exploring commercial adoption of this work in partnership with some Vancouver-based companies.

Direction 3: Java-script Based Web Applications' Reliability

Modern web applications have become pervasive and are fast 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 and maintenance. This interactivity is achieved by the use of client-side code written primarily in the JavaScript programming language, which is interpreted and executed within the web browser. JavaScript is a loosely typed, highly dynamic language, which places few restrictions on what programs can do. As a result, programs written in JavaScript are believed to be highly error prone, and there has been significant effort on designing alternatives to the JavaScript language e.g., DART from Google, TypeScript from Microsoft, and Flow from Facebook.

Together with my students and colleagues, I performed one of the first empirical studies on understanding the sources of errors in real-world JavaScript-based web applications [20]. Our study found that most errors actually do not have any effect on the application, and applications continue to execute satisfactorily despite encountering exceptions. Further, we found that most of the errors that matter for the application's correctness have to do with its interaction with the Document Object Model (DOM), a hierarchical and dynamic structure that is updated by the application to make changes in the displayed webpage [21]. Our study also found that DOM-related JavaScript errors take longer for developers to localize and fix. Our work thus overturns the popular notion that errors in web applications are due to the nature of the JavaScript language, and hence techniques that aim to replace or enhance JavaScript alone will not solve the problem.

In this work too, I applied the good enough dependability paradigm to selectively focus on DOM-related errors in web applications. Together with my students, I have built systems that can detect, localize, and repair DOM-related errors in web applications automatically [22][23][24]. We have also developed tools for programmers to better understand [25][26] and test web applications [27][28], as well methodologies for writing programs to avoid DOM-related JavaScript errors [29][30]. These papers have been published in conferences such as the IEEE/ACM International Conference on Software Engineering (ICSE), the premier-most venue for software engineering research. One of our papers at this conference received the ACM Distinguished paper award, an honour given to only nine of more than 500 submissions. We have also received a best paper nomination and a best paper runner up award at the IEEE International Conference on Software Testing (ICST). This research has been supported by companies such as Intel, Microsoft Research and SAP, and we are working with them to transition our research prototypes into their products.

Future Work

I believe the future will be shaped by the advent of massive amounts of computing, often in energy constrained, unreliable and insecure environments. Under such conditions, reliability, security and energy-efficiency become extremely important. In the future, I plan to extend the good enough dependability paradigm in three directions outlined below.

1. **Dependability for cloud computing**: Along with my students and collaborators, I recently studied failures in cloud computing clusters using the publicly available Google cluster workload traces. Our study, which was the first of its kind on public cloud data, found that failures in cloud applications exhibit regular patterns, and hence it is possible to predict the occurrence of failures much earlier to take corrective actions [31]. We further showed that it is possible to achieve significant resource savings through failure prediction for cloud jobs [32]. Going forward, I would like to explore failure mitigation strategies in the cloud through the good enough paradigm. For example, if we know that certain kinds of jobs are more prone to failures than others, perhaps we can execute them with increased error checking or monitoring to detect failures early. Another interesting direction would be to selectively duplicate the most failure prone jobs in the cloud to ensure that they complete, at the cost of higher resource utilization.

2. **Formal foundations:** I also plan to explore the formal foundations of the good enough dependability approach, and demonstrate that "good enough" systems can satisfy formal invariants. I have made forays in this direction in my DSN'08 paper where we used model checking to verify programs in the presence of soft errors [3], and in my CSF'11 paper where we used Hoare logic and theorem-proving to verify the security provided by the selective protection approach [17]. However, there is currently no unified approach to formally proving the adequacy of the protection provided by the good enough dependability paradigm for a wide variety of reliability and security issues. Such an approach will require fundamental innovations in both formal methods and computer systems design - I plan to explore both these directions.

3. **Evolutionary Approaches:** So far, my research has used deterministic approaches for applying the good enough paradigm to systems. For example, we use either automated techniques such as static and dynamic analysis (for reliability), or programmer-supplied annotations (for security) to designate which parts of an application need to be protected. In the future, I plan to leverage evolutionary approaches such as genetic programming to make these decisions based on the system's environment and conditions. For example, the same software system may have very different dependability requirements when deployed on space missions (where soft error rates are much higher) compared to terrestrial deployments. Using genetic programming, one can evolve the system towards an optimal state of resilience depending on the environment it is deployed in. Recent work has shown the feasibility of using evolutionary approaches for patching software bugs automatically [33][34]. However, there is no holistic method today to provide end-to-end system resilience from both hardware and software faults – this will be my research focus. I believe this is the future of computing and I would like to shape it through my research.

References

[1] Karthik Pattabiraman, Giacinto Paolo Saggese, Daniel Chen, Zbigniew Kalbarczyk and Ravishankar Iyer, *Automated Derivation of Application-specific Error Detectors using Dynamic Analysis*, IEEE Transactions on Dependable and Secure Computing (TDSC), 8(5), 2011.

[2] Karthik Pattabiraman, Zbigniew Kalbarczyk and Ravishankar Iyer, *Automated Derivation of Application-aware Error Detectors using Static Analysis,* IEEE Transactions on Dependable and Secure Computing (TDSC), 8(1), 2011.

[3] *Karthik Pattabiraman*, Nithin Nakka, Zbigniew Kalbarczyk and Ravishankar Iyer, *SymPLFIED: Symbolic Program Level Fault-Injection and Error Detection Framework,* Proceedings of the IEEE/IFIP International Conference on Dependable Systems and Networks (DSN), 2008, Anchorage, AK. (Acceptance Rate: 25%).

[4] Li, Man-Lap, Pradeep Ramachandran, Swarup K. Sahoo, Sarita V. Adve, Vikram S. Adve, and Yuanyuan Zhou. "Swat: An error resilient system."*Proceedings of SELSE* (2008).

[5] Feng, Shuguang, Shantanu Gupta, Amin Ansari, and Scott Mahlke. "Shoestring: probabilistic soft error reliability on the cheap." In *ACM SIGARCH Computer Architecture News*, vol. 38, no. 1, pp. 385-396. ACM, 2010.

[6] Galen Lyle, Shelley Chen, *Karthik Pattabiraman*, Zbigniew Kalbarczyk and Ravishankar Iyer, *An End-to-end Approach for the Automatic Derivation of Application-aware Error Detectors*, Proceedings of the IEEE/IFIP International Conference on Dependable Systems and Networks (DSN), Estoril, Portugal, 2009.

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

[8] Sampson, Adrian, Werner Dietl, Emily Fortuna, Danushen Gnanapragasam, Luis Ceze, and Dan Grossman. "EnerJ: Approximate data types for safe and general low-power computation." In *ACM SIGPLAN Notices*, vol. 46, no. 6, pp. 164-174. ACM, 2011.

[9] Carbin, Michael, Sasa Misailovic, and Martin C. Rinard. "Verifying quantitative reliability for programs that execute on unreliable hardware." In*ACM SIGPLAN Notices*, vol. 48, no. 10, pp. 33-52. ACM, 2013

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

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

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

[13] **Guanpeng Li, Qining Lu,** and Karthik Pattabiraman, *Fine Grained Characterization of Faults Causing Long Latency Crashes in Programs,* Proceedings of the IEEE/IFIP International Conference on Dependable Systems (DSN), 2015.

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

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

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

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

[18] **Farid Tabrizi** and Karthik Pattabiraman, *Flexible Intrusion Detection Systems for Memory-Constrained Embedded Systems,* Proceedings of the 11th European International Conference on Dependable Computing (EDCC), March 2015.

[19] Cole Schlesinger, Karthik Pattabiraman, Nikhil Swamy, David Walker, and Benjamin Zorn, *Modular Protections against Non-control Data Attacks,* Journal of Computer Security (JCS), 22(5): 699-742, 2014.

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

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

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

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

[24] **Frolin Ocariza,** Karthik Pattabiraman and Ali Mesbah, *Finding Inconsistencies in JavaScript MVC Applications*, Proceedings of the IEEE/ACM International Conference on Software Engineering (ICSE), 2015

[25] ***Saba Alimadi*, Sheldon Sequira,** Ali Mesbah and Karthik Pattabiraman, *Understanding JavaScript Event-Based Interactions,* Proceedings of the IEEE/ACM International Conference on Software Engineering (ICSE), 2014, Hyderabad, India

[26] **Saba Alimadadi,** Ali Mesbah and Karthik Pattabiraman, *Understanding Asynchronous Interactions in Full-Stack JavaScript,* To appear in the Proceedings of the IEEE/ACM International Conference on Software Engineering (ICSE), 2016.

[27] ***Shabnam Mirshokraie,*** Ali Mesbah and Karthik Pattabiraman, *Efficient JavaScript Mutation Testing*, Proceedings of the IEEE International Conference on Software Testing, Verification and Validation (ICST), 2013, Luxembourg.

[28] **Shabnam Mirshokraie,** Ali Mesbah and Karthik Pattabiraman, *JSEFT: Automated JavaScript Unit Test Generation,* Proceedings of the IEEE International Conference on Software Testing (ICST), 2015.

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

[30] **Kartik Bajaj,** Karthik Pattabiraman and Ali Mesbah, *Synthesizing Web Element Locators,* Proceedings of the IEEE/ACM International Conference on Automated Software Engineering (ASE), 2015.

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

[32] **Xin Chen,** Charng-da Lu and Karthik Pattabiraman, *Failure Prediction of Jobs in Compute Clouds: A Google Cluster Case Study,* International Workshop on Reliability and Security Data Analysis (RSDA), September 2014.

[33] Weimer, Westley, ThanhVu Nguyen, Claire Le Goues, and Stephanie Forrest. "Automatically finding patches using genetic programming." In *Proceedings of the 31st International Conference on Software Engineering*, pp. 364-374. IEEE Computer Society, 2009

[34] Weimer, Westley, Stephanie Forrest, Claire Le Goues, and ThanhVu Nguyen. "Automatic program repair with evolutionary computation."*Communications of the ACM* 53, no. 5 (2010): 109-116.