Automated Debugging and Optimization

Abhilash Jindal

My research vision is to *automate the "creative" task of debugging and optimization*– the task of finding bugs (or performance bottlenecks) in a software and making modifications to fix the software bugs (or improve software performance). To this end, I study program analysis and OS techniques to simplify the debugging and optimization efforts of programmers interacting with complex computer systems.

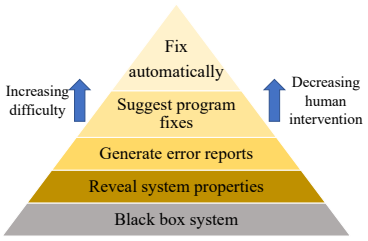

Figure 1: Automated debugging and optimization: Advanced tools reduce human intervention but are harder to build.

Debugging and optimization are largely ad-hoc manual processes taking up 35-75 percent of programmers' time costing more than $100B annually[1]. This process becomes further exacerbated in the modern programming paradigm where programmers stand on *the shoulder of giant software frameworks* to quickly program complex systems, but have a limited understanding of the intricacies of the underlying system. Simplifying this process will make developer productivity skyrocket and make complex computer systems more *accessible* to novice programmers. For example, empowered by powerful mobile frameworks and associated tooling, Masako Wakamiya, a 79-year old self-taught Japanese lady, was able to create a widely popular iOS game, *hinadan*, for the elderly. My research tackles challenges in making existing developers lot more productive and in making a wide range of complex systems more accessible so that many would-be developers like Masako can create socially beneficial applications.

The Sisyphean journey to full automation can be broken down into a few high-level system milestones illustrated in Figure 1. Each milestone reduces human effort but offer challenges of increasing difficulty:

- *Reveal system properties.* What are the right metrics to collect? How can these metrics be accurately collected with minimal monitoring overhead in production environments?
- *Generate error reports from runtime measurements or during static analysis.* What information should be surfaced in an error report to make it more actionable? How to assist developers to quickly arrive at a fix from an error report? How to generate errors with minimal false negatives and false positives?
- *Suggest program fixes.* How to bridge the gap between low-level error signatures and high-level program fixes? How to navigate the vast search space for potential fixes and optimizations?
- *Fix automatically.* How to create a repository of precise bug definitions? How to work with imprecise static analyses to generate precise fixes? How to reason about program's semantic properties from program syntax?

My thesis research [1] addressed some of these ongoing challenges in the context of mobile energy debugging and optimization. Due to the diversity of these challenges, my work is motivated by, and in-turn, motivates research in other areas, including algorithms, HCI, and programming languages. Figure 2 shows my research at a glance, grouped into simplifying *debugging specific bug categories* and *uncovering generic optimizations*. I have also made research contributions in creating *system-wide improvements* to address holistic system bottlenecks identified by my research. Below, I summarize my research followed by real-world impact and my future research plans.

Debugging specific bug categories

When system requirements change, systems undergo paradigm shifts that may require creating new abstractions. Such abstractions, if not designed correctly, seriously encumber programming, making writing correct programs difficult. I studied one such paradigm shift in the power management of smartphones which gives rise to a whole new class of bugs, *sleep disorder bugs*. These bugs plague the complete smartphone software stack: apps, framework, kernel, and device drivers.

[1] The Debugging Mindset, Devon H. O'Dell, ACM Queue, Volume 5, issue 1, 2017.

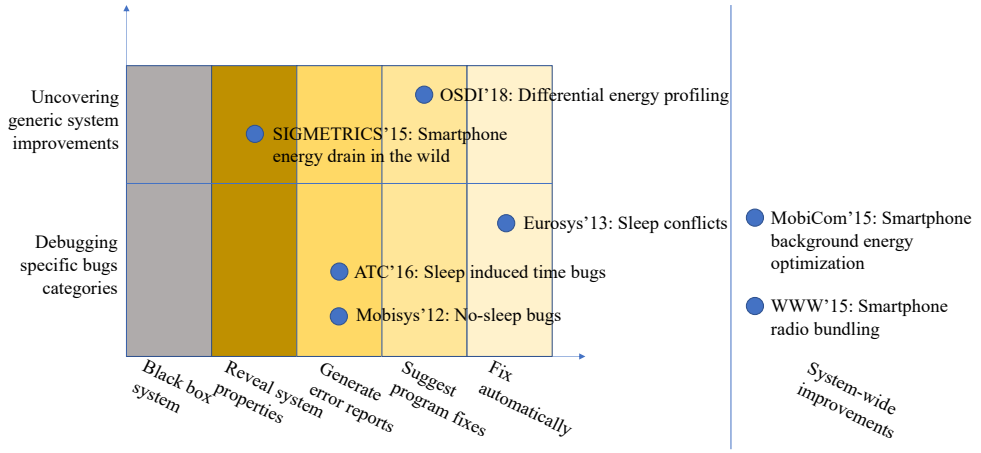

Figure 2: My thesis research at a glance. Mapping conference papers to the automated diagnosis space for *uncovering generic optimizations* and for *debugging specific bug categories*.

Rise of sleep disorder bugs. To save the crucial battery, smartphone OSes aggressively suspend the system on chip (SoC) after a brief period of user inactivity. However, this can hurt program correctness as the program may be in the middle of a *time-critical section* whose execution needs to be continuous, i.e., not disrupted by the system going to sleep.

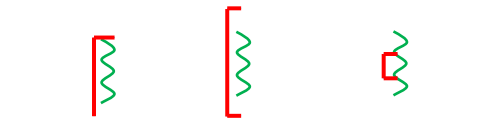

Figure 3: A taxonomy of sleep-disorder bugs. The green wiggles show the time-critical sections and the red bars show the code section across which the CPU is kept awake, by holding a wakelock.

To prevent phones from suddenly going to sleep during such time-critical sections, the OS provides explicit power control mechanisms— *wakelocks* with acquire and release APIs. This encumbers programming as now developers have to juggle these power control APIs along with the normal program logic to ensure the correct operation of their apps. Since developers typically have no training in doing so, they invariably make programming mistakes as evidenced by hundreds of such mistakes found in my research across the smartphone software stack. A direct consequence of such mistakes is that the CPU can go to sleep when it is supposed to stay awake, and vice versa. I term such programming mistakes that alter program semantics or cause unexpected battery drain due to smartphone sleep behavior as *sleep disorder bugs*.

Characterizing sleep disorder bugs. I conducted the first characterization study of time-critical sections to understand the properties of code sections whose execution needs to be continuous. I further developed a taxonomy of sleep disorder bugs as shown in Figure 3.

(1) *No-sleep bugs* happen when a wakelock is acquired correctly, e.g., to prevent the CPU from going to sleep, but is not released in some paths of program execution. The consequence is that the CPU cannot go to sleep, draining the phone battery indefinitely.

(2) *Under-sleep bugs* happen when wakelock APIs are properly matched, i.e., each wakelock acquire is later matched with a wakelock release, and hence the CPU can eventually go to sleep. However, the code section protected by the wakelock is larger than necessary keeping the CPU awake longer than necessary, wasting energy.

(3) *Over-sleep bugs* happen when the developer does not protect a time-critical section with wakelocks, creating the possibility of system suspend in the middle of the time-critical section. This type of bugs, unlike no-sleep and under-sleep bugs, can affect the correctness of the app.

The taxonomy was published in HotPower 2013 [2], workshop on power-aware computing colocated with SOSP 2013. This taxonomy enabled systematic treatment of sleep disorder bugs.

Treating sleep disorder bugs. Treating sleep disorder bugs requires us to automatically find time-critical sections, find wakelock-protected code sections where the CPU is kept awake, and then report mismatches between the two code sections. However, automatically finding time-critical sections, in general, is a hard problem since it requires guessing programmers' intentions from their programs' source code and runtime behaviour.

My research makes progress on this challenge by building specialized solutions that target individual classes of sleep disorder bugs. These bug classes contain hints in the program's source code or its runtime behaviour that enable finding time-critical sections and wakelock-protected code sections automatically.

Finding *no-sleep bugs* requires finding wakelock-protected code sections but does not necessitate finding time-critical sections. In [3], I built a static analysis tool based on reaching-definitions analysis that finds wakelock acquires that are never followed with a wakelock release on some program execution paths. The tool detects no-sleep bugs in the Android framework and apps.

In [4], I tackled a subclass of over-sleep bugs in device drivers called *sleep conflicts*. Sleep conflicts happen when a device driver cannot make progress in its execution, because of the CPU sleeep, to drive the power transition of its device back to its base power state. This leaves the device in a high power state, leaking battery. I built a runtime system that monitors the power state of all the devices by intercepting calls between the device driver and the hardware. This system automatically finds and mitigates sleep conflicts by ensuring devices power states at the time of system suspend.

In [5], I found another subclass of over-sleep bugs, *sleep-induced time bugs*, which manifest as logical errors from unexpected CPU sleep during the manipulation of time-related values. Here, the time-critical sections can be guessed because of well-defined time APIs accessed in the source code. I built a static analysis tool that uses use-def, def-use chains to report sleep-induced time bugs in the Linux kernel.

The tools collectively found hundreds of new bugs across all software layers including popular apps like Facebook, Android framework, Android kernel, and its device drivers. These works were published in Mobisys 2012 [3], Eurosys 2013 [4], and ATC 2016 [5], leading conferences in mobile systems and operating systems. Sleep disorder bugs are not a solved problem with new research continuing to evaluate better design of power-control APIs[2] and are likely to further increase in prevalence with more low-power devices making their way into consumer lives.

Uncovering generic optimizations

While effective, the systems built in the previous approach can only find bugs that belong to specific bug categories. The holy grail of automated debugging and optimization is to build general purpose tools that can uncover arbitrary system improvements. I studied this problem in the context of improving battery drain of smartphones. I did first large-scale measurements to identify holistic bottlenecks, proposed practical solutions to address major bottlenecks, and researched ways to suggest program improvements to developers for reducing their program's battery drain.

Measuring battery drain behavior of smartphones to identify holistic bottlenecks. I believe it is imperative to get a holistic view of the system's performance landscape before diving into its optimization. Although measuring each system comes with its own set of unique challenges, they share some common hurdles that need to be overcome: 1. developing a low-overhead and accurate logging mechanism to measure system's performance metrics and 2. incentivizing a large number of users for data collection.

To overcome these problems in the scope of measuring how smartphones drain the battery, I implemented an Android app called Estar. Existing power accounting mechanisms in the literature required collecting extensive triggers, such as packet-level trace, that could not be collected in an in-the-wild app as they require root-level privileges. In Estar, I developed accurate power accounting mechanisms and a sophisticated logging mechanism with $\leq$0.6% energy overhead. Estar incentivized users by showing them an *"energy-star"* rating before they install any new app from Google Play. I generated this energy-star rating by crowdsourcing data from all our users. Without any marketing expenditure, Estar quickly became the top-trending app on Google Play, providing data from over 100,000 users from 56 countries. This measurement study was published in SIGMETRICS 2015 [6], the top publication venue for system measurement research.

Improving system-wide bottlenecks. The measurement study highlighted that the background apps and services during screen-off intervals drain 23% of the total phone battery drain on average. Driven by this, I closely examined their behaviour and found a surpising inefficiency – many apps were draining battery during the screen-off interval, downloading content from remote servers, despite not

[2] A Case for Lease-Based, Utilitarian Resource Management on Mobile Devices, ASPLOS 2019.

being opened by the user! To formulate this systemic inefficiency, I defined a mathematical formulation, *background-foreground correlation (BFC)*, to measure the usefulness of app background activities for each app. I further designed a screen-off energy optimizer on Android called HUSH that monitors the BFC of all apps on a phone online and automatically identifies and suppresses app background activities during screen-off intervals that are not useful to the user. In doing so, HUSH saves the screen-off energy of smartphones by 15.7% on average with minimal impact on the user experience with the apps. These results were published in Mobicom 2015 [7], the top conference on mobile computing.

Suggesting program improvements to developers for improving performance. Over the decades, the systems community has created many interesting profilers for automatically or semi-automatically narrowing the program scope wherein making a modification is likely to enhance software performance. However, these profilers still fall short of the holy grail as after being presented with performance hotspots, developers do not have any guidance on how to proceed with the remaining optimization task: 1. Is there a more efficient implementation? 2. How to come up with a more efficient implementation?

My research made progress on these hard challenges in the context of mobile energy profilers by making the following key observations about mobile app development: (1) For every popular app in the app market, there are a few dozen competing apps that implement similar app features, (2) Similar apps can differ significantly in energy drain in performing similar app functions, (3) Mobile apps heavily use common framework services, such as the Android framework, and these framework code segments show up in energy hotspots lot more often than the code written by the app programmers.

I leveraged these insights in DIFFPROF, a *differential energy profiling* tool for the energy optimization of smartphone apps. DIFFPROF employed a novel tree-matching algorithm for comparing energy profiler outputs of two similar apps and found 12 energy improvements in popular apps. The case studies showcased that DIFFPROF provides developers with actionable diagnosis beyond a traditional profiler: it identifies non-essential (unmatched or extra) and known-to-be inefficient (matched) tasks, and further allow developers to quickly understand the reasons and develop fixes for the energy difference from a competing app with minor manual debugging efforts. The research results were published in a top venue in the systems field (OSDI 2018) [8] and led to an NSF grant 1718854 of $475k.

Research impact

I get excited about creating systems that directly improve people's lives and are grounded in strong foundations. In addition to simplifying programmers' debugging and optimization efforts, my research has made significant advances towards extending the limited battery life of smartphones making societal impact on people across the economic spectrum.

- The papers on sleep disorder bugs together laid the foundation for systematic treatment of misuse of power control APIs. The tools we built in the papers collectively **found hundreds of new bugs** across all software layers including popular apps like Facebook, Android framework, Android kernel, and its device drivers. The papers inspired a large number of follow-on work by the research community with **over 350 citations** (Google Scholar) and were widely covered in the media. The work started a discussion among the major players of the mobile ecosystem[3]about the misuse of wakelock APIs and further inspired several development tools for detecting these bugs such as **Wakelock and WakelockTimeout lint checks in Android Studio** and **reporting stuck wakelocks to app developers in Android Vitals**.

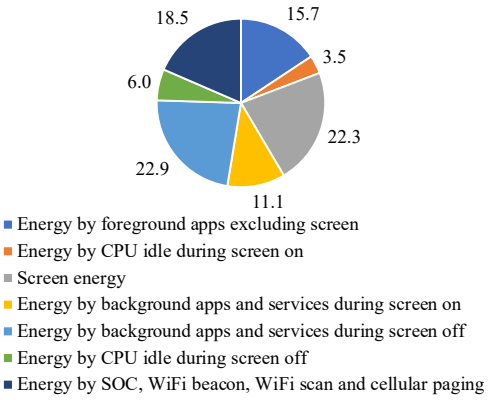

Figure 4: Average daily battery drain breakdown across 2000 users from [7].

- Our measurement study results (such as shown in Figure 4) drew several implications for major players in the mobile ecosystem: the phone vendors, SOC vendors, cellular carriers, and app

[3]Wakelocks: Detect No-Sleep Issues in Android Applications. Intel Developer Zone, 2013.

developers on a better system, network, and app design to extend battery life. The study **got widely covered in the media**[4]. The high screen energy identified by our measurement study as a major contributor of daily battery drain is now being targeted by the **system-wide dark mode** features in the latest Android, iOS and macOS versions.

- We published the HUSH source-code in Github. HUSH repository has been **forked over 50 times** by the developer community. Android team released ***app-standby*** **and** ***Doze*** **features very similar to HUSH** in the Marshmallow version.
- Differential energy profiling became the **key feature in Eagle Debugger**[5], an energy diagnostics solution for mobile apps, developed by the startup I co-founded, Mobile Enerlytics.

Future research

Debugging and optimization is universal to all computer systems permeating the entire software industry. I am excited to build upon my experience to continue to steadily climb the automation pyramid (Figure 1). In the longer term, I wish to develop general guiding principles for any program improvement tool and build robust theories around decidability of automated fixing of programs. In the near future, I plan to work on following problems:

[Reveal system properties] Fine-grained power consumption analysis of large-scale distributed systems. Data centers already consume 2% of the global energy consumption and due to the explosive growth of machine learning they are predicted to consume up to 8% of the projected global demand by 2030[6]. I believe a thorough understanding of how a data center consumes power as a whole and how individual applications and scheduling strategies contribute to the overall power consumption are top-priority challenges. It is hard to model and measure data center power consumption for the following reasons– (1) the level of support given by most modern hardware systems for measuring power consumption is insufficient, (2) power models are not portable across heterogenous systems, and (3) power model accuracy rapidly degrades across different workloads and different data center deployments. I plan to investigate a self-generating and self-correcting power modelling methodology that correlates individual system events with aggregated power consumption. Once a robust power model is created, it can be leveraged to create a fine-grained energy profiler that reveals the energy consumption behaviour of complex distributed applications and scheduling strategies. Such observability of data center power consumption shall uncover previously undiscovered power optimization opportunities.

[Generate error reports] Root cause analysis of performance variations. System performance can be adversely affected by a modification in system's source code, by variations in execution such as taking two different A/B test execution branches, or by myriad of environmental factors such as changing network conditions, dynamic scheduling decisions, especially in the big.LITTLE architecture, and memory pressure at the time of code execution. Separating these root causes is imperative for generating actionable error reports in performance monitoring setups such as in-house regression testing, and in-the-wild performance monitoring. I plan to explore data science and machine learning based approaches, such as clustering, classification and hypothesis testing, to train models that are able to identify why system performance metrics get affected. Using the trained models, I would like to further understand and enumerate all root causes that impact performance of a given system. This will guide the next step of defining actionable error reporting mechanisms that can quickly guide developers toward fixes.

[Suggest program fixes] Deep machine understanding of APIs to suggest program fixes. Modern applications are written using specialized programming frameworks such as ReactJS for frontend development, TensorFlow for developing ML models, Spring for backend development, etc. As these framework services and libraries mature, they continue to offer higher-level intricate functionalities to programmers via flexible APIs. Abundance of multiple applications reusing the same APIs in slightly different ways to implement similar functionalities has opened up new exciting research possibilities for suggesting high-fidelity program fixes and improvements through deep machine understanding of APIs.

[4] Smartphone Battery Drains a Lot Even with Dark Screen. Scientific American, 2015.
[5] Eagle Debugger Energy Comparison (https://www.youtube.com/watch?v=5FF72KaKILg).
[6] Nature: How to stop data centres from gobbling up the world's electricity?, September 2018.

I plan to extend my recent work [8] and explore how traditional tools such as profilers can embrace this modern programming paradigm to directly suggest program fixes.

Making progress in above areas requires multi-disciplinary research with hard problems needing to be solved in diverse areas of computer science and engineering. I plan to work closely within our department with computer theorists, hardware engineers, PL and ML experts where their domain expertise combined with my systems building ability create highly impactful research. Outside the department, I will collaborate with Electrical and Mechanical engineering colleagues to build accurate tools to show a holistic view of and debug issues with the power consumed by data centers. I further plan to use my customer discovery experience from my startup to build appropriate research processes to engage industry leaders to thrusts these advanced systems toward broader adoption.

References

[1] **Abhilash Jindal**. "Towards Automated Energy Debugging on Smartphones". PhD thesis. Purdue University, Aug. 2017.

[2] **Abhilash Jindal**, Abhinav Pathak, Y. Charlie Hu, and Samuel P. Midkiff. "On death, taxes, and sleep disorder bugs in smartphones". In: *Proceedings of the Workshop on Power-Aware Computing and Systems (HotPower)*. ACM. 2013, p. 1.

[3] Abhinav Pathak, **Abhilash Jindal**, Y. Charlie Hu, and Samuel P. Midkiff. "What is keeping my phone awake?: Characterizing and detecting no-sleep energy bugs in smartphone apps". In: *Proceedings of the 10th international conference on Mobile systems, applications, and services (MobiSys)*. ACM. 2012, pp. 267–280.

[4] **Abhilash Jindal**, Abhinav Pathak, Y. Charlie Hu, and Samuel P. Midkiff. "Hypnos: understanding and treating sleep conflicts in smartphones". In: *Proceedings of the 8th ACM European Conference on Computer Systems (EuroSys)*. ACM. 2013, pp. 253–266.

[5] **Abhilash Jindal**, Prahlad Joshi, Y. Charlie Hu, and Samuel P. Midkiff. "Unsafe time handling in smartphones". In: *2016 USENIX Annual Technical Conference (ATC)*. USENIX Association. 2016, pp. 115–127.

[6] Xiaomeng Chen, Ning Ding, **Abhilash Jindal**, Y. Charlie Hu, Maruti Gupta, and Rath Vannithamby. "Smartphone energy drain in the wild: Analysis and implications". In: *Proceedings of the 2015 ACM SIGMETRICS International Conference on Measurement and Modeling of Computer Systems*. ACM. 2015, pp. 151–164.

[7] Xiaomeng Chen, **Abhilash Jindal**, Ning Ding, Y. Charlie Hu, Maruti Gupta, and Rath Vannithamby. "Smartphone background activities in the wild: Origin, energy drain, and optimization". In: *Proceedings of the 21st Annual International Conference on Mobile Computing and Networking (MobiCom)*. ACM. 2015, pp. 40–52.

[8] **Abhilash Jindal** and Y. Charlie Hu. "Differential energy profiling: energy optimization via diffing similar apps". In: *Proceedings of the 12th USENIX conference on Operating Systems Design and Implementation (OSDI)*. USENIX Association. 2018, pp. 511–526.

[9] Ana Nika, Yibo Zhu, Ning Ding, **Abhilash Jindal**, Y. Charlie Hu, Xia Zhou, Ben Y. Zhao, and Haitao Zheng. "Energy and performance of smartphone radio bundling in outdoor environments". In: *Proceedings of the 24th International Conference on World Wide Web (WWW)*. International World Wide Web Conferences Steering Committee. 2015, pp. 809–819.

[10] Faez Ahmed, **Abhilash Jindal**, and Kalyanmoy Deb. "Cricket team selection using evolutionary multi-objective optimization". In: *International Conference on Swarm, Evolutionary, and Memetic Computing (SEMMCO)*. Springer. 2011, pp. 71–78.

[11] Faez Ahmed, Kalyanmoy Deb, and **Abhilash Jindal**. "Multi-objective optimization and decision making approaches to cricket team selection". In: *Applied Soft Computing* 13.1 (2013), pp. 402–414.

[12] Xiaomeng Chen, **Abhilash Jindal**, and Y. Charlie Hu. "How much energy can we save from prefetching ads?: energy drain analysis of top 100 apps". In: *Proceedings of the Workshop on Power-Aware Computing and Systems (HotPower)*. ACM. 2013, p. 3.

[13] Mark S. Drew, Graham D. Finlayson, and **Abhilash Jindal**. "Colour image compression by grey to colour conversion". In: *Computational Imaging IX*. Vol. 7873. International Society for Optics and Photonics. 2011, 78730Z.