Building Systems for Mobile & Cloud Applications

Irene Zhang - Research Statement

The proliferation of datacenters, smartphones, personal sensing and tracking devices, and home automation products is fundamentally changing the applications we interact with daily. Modern applications are no longer limited to a single desktop computer but now commonly span many machines and devices. In my research, I identify new applications and their problems, then build new operating and distributed systems to meet their needs.

Throughout my career, I have built systems that run at every level of the systems stack. I began my research career working on a new processor architecture [2], while my thesis focuses on large-scale, distributed systems that span mobile devices and the cloud [19, 13, 16]. In between, I have worked on virtual machine checkpointing [14, 15], wide-area file systems [12], and datacenter caching [11].

I use this experience to design new systems and improve existing ones. For example, I used an old idea – memory-mapped storage – to create a new reactive data management system called Diamond [16]. I observed the power of interposition in virtualization and applied it to an extensible runtime system for distributed applications in the Sapphire project [19]. I used the end-to-end principle to revolutionize distributed transactional storage by building strong transactional guarantees atop a replication layer with no consistency guarantees in TAPIR [17]. This practice of refining or applying core operating systems principles to new and existing systems characterizes my research approach.

1 Thesis Work: New Systems for Mobile/Cloud Applications

My PhD thesis work is the first to identify – and design systems to support – *mobile/cloud applications*: applications that split their logic across mobile devices and cloud servers with significant communication and coordination between the two. Mobile/cloud applications are now commonplace; millions of users log on to Facebook, Google Docs, and Twitter every day. Despite their ubiquity, systems to help programmers build these applications have not kept pace. Lacking end-to-end systems support, today's mobile/cloud applications must take responsibility for traditional OS tasks (e.g., managing their execution environment, security, and storage and memory). As a result, application programmers must become distributed systems and operating systems experts. For my thesis, I designed and built three end-to-end systems for mobile/cloud applications to eliminate these traditional OS tasks and simplify application development.

Diamond simplifies mobile/cloud application storage. Mobile/cloud applications are increasingly *reactive*: they automatically propagate updates across devices and the cloud without requiring users to save, reload or explicitly exchange data. Some examples include social networks (e.g., Facebook, Twitter), cloud-based desktop applications (e.g., Office 365), collaboration applications (e.g., Trello) and multi-player games (e.g., Words With Friends). Reactive applications face the challenging task of automatically synchronizing copies of shared application data across many distributed processes. More complicated still, these processes can fail, become unreachable, and make concurrent updates. As a result, programmers must implement *distributed consistency protocols* in their applications. They typically eschew strong consistency due to the difficulty of implementing the protocols correctly and with good performance. Unfortunately, weak consistency results in unpredictable race conditions and hard-to-diagnose bugs in the applications. Users are becoming increasingly intolerant of these inconsistencies because mobile/cloud applications now commonly deal with real money (e.g., social networks ads and in-game purchases are billion dollar markets).

Diamond [16] is the first *data management system* that manages distributed memory and storage for reactive, mobile/cloud applications. It introduces a new abstraction, called *reactive data map*, that lets programmers bind application variables to Diamond's cloud storage. Once bound, Diamond automatically propagates updates from application memory to storage with strong consistency and fault-tolerance guarantees. In this way, Diamond provides persistent and flexible distributed shared memory. We designed, implemented, and released Diamond as open-source software on Github (https://github.com/UWSysLab/diamond). Our experiments show that Diamond reduces application code size – by 13% to 33% – and *transparently* provides stronger consistency guarantees for a wide range of applications. More importantly, Diamond eliminates the need for programmers to implement distributed protocols for memory management and storage.

Sapphire simplies mobile/cloud application execution. Mobile/cloud applications span diverse execution environments; the cloud environment provides fast computation with excellent network connectivity, while mobile devices provide limited computation with variable network connectivity. This dramatic range of capabilities forces programmers to make *deployment decisions*: they must decide how to partition application components across execution environments and how to communicate and coordinate between those components. These decisions are: (1) *application-dependent* (e.g., a computation-heavy component may be better deployed on a cloud server, while a user-facing one may be better deployed on a mobile device) and (2) are *changing rapidly* due to changing hardware and application requirements. Currently, application programmers make and implement these decisions, which force them to confront complex distributed systems and operating systems issues, such as where to place processes, where to cache data, and how to reliably communicate. Worse, deployment code is strewn throughout applications, which makes it difficult to modify when deployment decisions change.

Sapphire [19] is the first *deployment management system* that manages distributed execution environments and communication for mobile/cloud applications. Its novel system architecture separates deployment decisions from implementation. Sapphire's architecture puts common low-level mechanisms (e.g., best-effort communication) in a *Deployment Kernel* (DK), separate from high-level deployment patterns (e.g., client-side caching and code offloading), which are implemented in pluggable *Deployment Managers* (DMs). To deploy a Sapphire application, programmers divide application functions into distributed *Sapphire Objects* (SOs) and then choose from a library of DMs to create a custom execution environment for their application. DMs interpose on operations to SOs in order to implement extensible functionality, such as layering at-most-once communication atop best-effort communication. We built Sapphire for Android applications and released it with a library of 26 DMs on Github (https://github.com/UWSysLab/Sapphire). Our experiments demonstrate that Sapphire both provides a general-purpose, end-to-end execution environment for mobile/cloud applications *and* supports a wide range of applications with different deployment needs. Using Sapphire, programmers need not implement deployment decisions and can easily change them. For example, we changed a server-based application to a peer-to-peer application by changing a *single line of code*.

Agate simplifies mobile/cloud application security. Mobile/cloud applications are *sharing-oriented*: mobile devices collect data from user interactions and sensors, while applications distribute this data through the cloud to other devices and users. As a result, protecting sensitive user data becomes an important and difficult task. Currently, mobile operating systems (e.g., Android) limit which applications can access sensitive data; however, once an application has access, users must trust it to correctly enforce their preferred policies. For example, Alice can give her exercise app access to her GPS, but she must trust that it will share her GPS location only with the users she selects. This places a heavy burden on application programmers to correctly enforce user policies and requires users to trust every application they use.

Agate [13] is the first *privacy management system* that enforces end-to-end protection for sensitive user data in sharing-focused, mobile/cloud applications. It provides both *access control* (where users control

which applications can access their data) and *flow control* (where users control with whom applications can share that data). To achieve this dual control, Agate uses information flow control (IFC) instead of traditional isolation mechanisms, like sandboxing. While IFC has been heavily researched in the context of giving programmers more control over privacy, it has never been previously used to give users control over their data due to the complex tagging mechanisms. Agate's key insight is to modify the mobile OS to automatically tag sensitive user data, eliminating the need for users to understand IFC or trust application programmers to enforce privacy on their behalf. We implemented Agate for Android and made it available on Github (https://github.com/SapphireAgate). Our experiments show that Agate supports a range of sharing policies, from ensuring that a Twitter clone can share users' photos only with their friends to ensuring that a chat log remains private to members of the chat room. Agate eliminates 19 of the top 20 web and mobile security risks identified by the Open Web Application Security Project [8, 9]. Importantly, users can rely on Agate to enforce their privacy policies without trusting applications or their programmers.

Summary. The complexity of application programming continues to increase: applications span billions of mobile devices in the hands of users and millions of servers in hundreds of datacenters around the world. In my research, I seek to build powerful systems that simplify the development of new applications. My thesis work thus has a common thread: building novel, general-purpose systems that reduce the responsibility now on application programmers' shoulders. The current situation is untenable; application programmers today must be experts in distributed systems, operating systems and security. We can and must develop systems to reduce the expertise needed to implement these everyday applications.

2 Distributed Systems for Cloud Applications

In addition to *building new systems* for mobile/cloud applications, I also *rearchitect* existing systems to better suit today's applications. The protocols and mechanisms that underlie popular cloud systems were designed long before these applications existed; as a result, they are general-purpose but inefficient. My primary focus in this area is TAPIR [17]; however, during my PhD, I also collaborated on the Arrakis [10] and IPA [4] projects. TAPIR's key contribution is to replace the general-purpose Paxos protocol with a new replication protocol specifically designed for transactional storage systems. As a result, we significantly reduced the cost of distributed transactions in a replicated storage system. Arrakis uses the hardware virtualization capabilities of new I/O devices to replace the general-purpose kernel-level I/O stack with customized mechanisms for each application, significantly reducing I/O processing overhead. IPA [4] replaces the general-purpose key-value interface with a data structure interface to bring a more principled approach to weak consistency.

TAPIR optimizes replicated, transactional storage. Transactional storage systems typically layer distributed protocols to achieve their guarantees. For example, Spanner [1] layers a distributed transaction protocol – consisting of Two-Phase Commit for atomicity and Strict Two-Phase Locking for isolation – atop a distributed replication protocol (e.g., Paxos) for fault-tolerance and consistency. The standard model for replication protocols today is either state machine replication or an ordered log, both of which impose a strict ordering of operations across replicas. We analyzed the replication layers in several systems and found that this model causes replication protocols to duplicate the guarantees of distributed transaction protocols, leading to over-coordination and lost performance [18].

This observation led us to design a new replication protocol, *inconsistent replication* (IR), that offers replication without strict operation ordering. Instead of an ordered operation log, IR provides an unordered operation set. Using it, we co-designed the Transactional Application Protocol for Inconsistent Replication (TAPIR) [17] to efficiently enforce linearizable transaction ordering using a completely unordered replication layer. We implemented TAPIR and several comparison systems and put them on Github

(https://github.com/UWSysLab/tapir). Our experiments demonstrated that, by efficiently leveraging IR's weak guarantees, TAPIR halves the commit latency and triples the throughput of conventional layered protocols. Other researchers have subsequently applied TAPIR's insights to highly contended workloads [7] and read-heavy datacenter workloads [6].

3 Future Directions

My research thus far opens many promising new directions for future work. I will continue to anticipate application and hardware trends and design systems that simplify application development.

Redesigning mobile and cloud operating systems for distributed applications. One direction I find especially exciting is redesigning existing operating systems for cloud servers and mobile devices. The rapid turnover of mobile devices and the centralized management of datacenters now makes possible the large-scale redesign of fundamental operating systems mechanisms. Operating systems running on both cloud servers and mobile devices share a key characteristic: their processes are *a small part of a larger distributed application*. This property has significant implications for every aspect of OS design – including scheduling, failure handling, and storage – because the OS no longer completely controls the entire application. In fact, the OS has become a less significant component of the stack both vertically (given the many libraries and runtimes layered on top of the OS) and horizontally (given the many other operating systems on machines running application processes). Thus, the OS must cooperate – not conflict – with these other layers and nodes. In general, I am interested in revisiting classic OS abstractions with an eye to both leveraging new hardware capabilities (e.g., NVRAM) and meeting new distributed application needs.

Leveraging machine learning and mobile sensors in OS scheduling. My previous work also presents a promising avenue for further research in process scheduling and data placement. Sapphire and Diamond decouple execution and data management, respectively, from mobile/cloud applications, making it possible to implement more complex scheduling policies. Building on this, my ultimate research goal is to predict application behavior and perform scheduling without programmer assistance. This is a formidable problem for mobile/cloud applications, where a wrong scheduling decision can cost seconds in latency and cause frustration for users. However, predictive algorithms have advanced significantly with machine learning, and mobile devices collect huge amounts of information that can improve decisions (e.g., a user's location, current activity, etc. can predict what data the user will access). Thus, I believe that even this ambitious goal is now within our reach.

Designing new distributed systems for research applications. Many computer science researchers are now building distributed applications as part of their projects. For example, GraphLab [5] grew from the need for machine learning researchers to run distributed ML algorithms. Sapphire can be easily repurposed into a research tool for building flexible platforms for research on heterogeneous architectures or mobile offloading. Diamond shows promise as a general-purpose, flexible shared memory system; for example, one compelling application is communication between cooperating robots. Robotics researchers are building complex, distributed algorithms for robots to communicate. These algorithms present a challenging distributed application because they require low latency communication and real-time scheduling. They are currently built in a message-passing style using ROS [3]; however, a system like Diamond, which provides distributed shared application state, could significantly reduce implementation complexity. In general, I am interested in building general-purpose programming systems to meet the needs of researchers with interesting distributed applications. My experience in both operating system and distributed systems ideally positions me to address the challenges that a wide range of applications will face in the future.

References

[1] James C. Corbett et al. Spanner: Google's globally-distributed database. In *Proceedings of the USENIX Symposium on Operating Systems Design and Implementation (OSDI)*, 2012.

[2] Jack B. Dennis. Fresh Breeze: a multiprocessor chip architecture guided by modular programming principles. *SIGARCH Comput. Archit. News*, 31(1), March 2003.

[3] Lee Garber. Robot OS: a new day for robot design. *Computer*, 46(12):16–20, 2013.

[4] Brandon Holt, James Bornholt, **Irene Zhang**, Dan R. K. Ports, Mark Oskin, and Luis Ceze. Disciplined inconsistency. In *Proceedings of the ACM Symposium on Cloud Computing (SoCC)*, 10 2016.

[5] Yucheng Low, Danny Bickson, Joseph Gonzalez, Carlos Guestrin, Aapo Kyrola, and Joseph M Hellerstein. Distributed GraphLab: a framework for machine learning and data mining in the cloud. *vldb*, 5(8):716–727, 2012.

[6] Pulkit A. Misra, Jeffrey S Chase, Johannes Gehrke, and Alvin R. Lebeck. Enabling lightweight transactional persistent memory with precision time. In *Proc. of the ACM International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS)*, 2017.

[7] Shuai Mu, Lamont Nelson, Wyatt Lloyd, and Jinyang Li. Consolidating concurrency control and consensus for commits under conflicts. In *Proceedings of the USENIX Symposium on Operating Systems Design and Implementation (OSDI)*, 2016.

[8] OWASP top 10 application security risks 2013, 2013. https://www.owasp.org/index.php/Top10.

[9] OWASP top 10 mobile risks 2014, 2014. https://www.owasp.org/index.php/Projects/OWASP_Mobile_Security_Project_-_Top_Ten_Mobile_Risks.

[10] Simon Peter, Jialin Li, **Irene Zhang**, Dan R. K. Ports, Douglas Woos, Arvind Krishnamurthy, Thomas Anderson, and Timothy Roscoe. Arrakis: The operating system is the control plane. In *Proceedings of the USENIX Symposium on Operating Systems Design and Implementation (OSDI)*, 10 2014. Best Paper Award.

[11] Dan R. K. Ports, Austin T Clements, **Irene Zhang**, Samuel Madden, and Barbara Liskov. Transactional consistency and automatic management in an application data cache. In *Proceedings of the USENIX Symposium on Operating Systems Design and Implementation (OSDI)*, 10 2010.

[12] Jeremy Stribling, Yair Sovran, **Irene Zhang**, Xavid Pretzer, Jinyang Li, M. Frans Kaashoek, and Robert Morris. Flexible, wide-area storage for distributed systems with WheelFS. In *Proceedings of the USENIX Symposium on Networked Systems Design and Implementation (NSDI)*, 4 2009.

[13] Adriana Szekeres, **Irene Zhang**, Katelin Bailey, Isaac Ackerman, Haichen Shen, Franziska Roesner, Dan R. K. Ports, Arvind Krishnamurthy, and Henry M. Levy. Enforcing privacy policies on mobile/cloud applications with the Agate privacy-preserving platform. In submission, 2016.

[14] **Irene Zhang**, Tyler Denniston, Yury Baskakov, and Alex Garthwaite. Optimizing VM checkpointing for restore performance in VMware ESXi. In *Proceedings of the USENIX Annual Technical Conference (ATC)*, 6 2013.

[15] **Irene Zhang**, Alex Garthwaite, Yury Baskakov, and Kenneth C. Barr. Fast restore of checkpointed memory using working set estimation. In *Proceedings of the International Conference on Virtual Execution Environments (VEE)*, 3 2011.

[16] **Irene Zhang**, Niel Lebeck, Pedro Fonseca, Brandon Holt, Raymond Cheng, Ariadna Norberg, Arvind Krishnamurthy, and Henry M. Levy. Automating data management for wide-area, reactive applications. In *Proceedings of the USENIX Symposium on Operating Systems Design and Implementation (OSDI)*, 11 2016.

[17] **Irene Zhang**, Naveen Kr. Sharma, Adriana Szekeres, Arvind Krishnamurhty, and Dan R. K. Ports. Building consistent transactions with inconsistent replication. In *Proceedings of the ACM Symposium on Operating Systems Principles (SOSP)*, 10 2015.

[18] **Irene Zhang**, Naveen Kr. Sharma, Adriana Szekeres, Dan R. K. Ports, and Arvind Krishnamurthy. When is operation ordering required in replicated transactional storage? *IEEE Data Engineering Bulletin*, 3 2016.

[19] **Irene Zhang**, Adriana Szekeres, Dana Van Aken, Isaac Ackerman, Steven D. Gribble, Arvind Krishnamurthy, and Henry M. Levy. Customizable and extensible deployment for mobile/cloud applications. In *Proceedings of the USENIX Symposium on Operating Systems Design and Implementation (OSDI)*, 10 2014.