Can Deterministic Replay be an Enabling Tool for Mobile Computing?

Jason Flinn and Z. Morley Mao
University of Michigan

1. ABSTRACT

Deterministic record and replay is fast becoming a vital technology in desktop and server computing environments. Yet, the applicability of this technology to computation run on small, mobile devices such as cell phones has not yet been explored. We argue that there are several potential uses of replay that are especially beneficial when applied to mobile phones: dual execution on cloud or cloudlet computers to reduce latency and possibly phone energy use, operation shipping for file synchronization, and offloading of expensive security and reliability checks to remote servers. In this paper, we explore these potential uses, as well as some of the unique challenges posed by implementing replay on phones.

2. INTRODUCTION

Deterministic replay has become an important foundational technology in desktop and server computing environments. A deterministic replay system provides DVR-like functionality, in which an execution of a hardware or software system is recorded so that an identical execution can later be replayed on demand. The ability to faithfully reproduce an execution has proven useful in many areas, including debugging [22, 33, 36], fault tolerance [4], computer forensics [11], dynamic analysis [7, 26], auditing [16], and workload capture [24, 39]. As a consequence, commercial products such as VMware Workstation currently include support for deterministic record and replay [39].

Deterministic replay systems work by logging all non-deterministic events during a recording phase, then reproducing these events during a replay phase. On uniprocessors, non-deterministic events (e.g., I/O, scheduling interrupts, and user interaction) occur relatively infrequently, so logging and replaying them incurs little overhead. One can guarantee that any subsequent execution of a program will produce identical results by starting from the same initial state (e.g., the executable image) and supplying the same results of non-deterministic events in the order that they occurred during the original execution.

HotMobile '11 Phoenix, AZ, USA

In this paper, we argue that deterministic replay can also serve as an important foundational technology for building software systems that target small, mobile devices such as cell phones. While several of the use cases we list above apply equally well to mobile computing systems as they do to desktop and server systems, there are several additional use cases that are unique to mobile computing.

First, deterministic replay provides a new way to leverage cloud and cloudlet [32] servers to reduce the latency of computation. Rather than guessing whether local or remote execution will yield faster response time, a mobile phone can run important computation at both locations and use the first response it receives. Whereas current function-shipping techniques can only show performance benefits when offloading coarse-grained computation due to the overhead of state synchronization, replay can potentially improve performance for even fine-grained computation because synchronous computation is replaced by asynchronous log propagation of inputs and other non-deterministic events.

Second, deterministic replay can reduce the cost of data synchronization. Prior work has observed that one can save bandwidth and energy for operations that modify a large amount of data by shipping a description of the computation to an entity with whom one wishes to synchronize data rather than the data itself. That entity can then replicate the computation to produce an identical modification to the data. A weakness of prior techniques has been the possibility of divergence due to non-deterministic application behavior. Deterministic replay guarantees that such divergence never occurs.

Third, deterministic replay provides a mechanism to leverage cloud and cloudlet servers to enhance the security and reliability of mobile computation. For state-based operations such as on-access virus scanning, one can simply ship a compact representation of the state to be checked to a server and wait for a result [27]. But other checks such as taint tracking, race detection, deadlock avoidance, and anomaly detection require that dynamic analysis be applied to an actual execution of the application being checked. Deterministic replay can make such checks possible by allowing a server to asynchronously validate an execution identical to the one being performed on the phone. Because of its greater computational resources, the server can employ heavyweight checks and still execute the original application at the same relative speed as the phone.

In the rest of this paper, we outline some of the challenges and opportunities for implementing deterministic replay in mobile computing environments. We first provide

background on how deterministic replay systems operate. Section 4 then explores three potential use cases for mobile replay in more detail. Section 5 outlines some of the unique challenges faced when deploying deterministic replay on mobile systems, and Section 6 concludes.

3. DETERMINISTIC REPLAY

The goal of deterministic replay is to capture an execution of a computer system so that the execution can subsequently be replayed. The success of replay is predicated on the observation that the bulk of a computer system's execution is deterministic. Thus, one need only log the small set of non-deterministic inputs and operations in order to capture an execution — these events happen at relatively low frequencies. When replaying a recorded execution, the non-deterministic operations are not re-executed. Instead, the results of those operations from the recorded execution are reproduced from the log and supplied to the replayed execution. This guarantees that the replayed execution will produce results identical to those produced by the original, recorded execution.

Record and replay can be implemented at several different layers of the computer system, and the specific set of non-deterministic events that must be logged depends on which layer is chosen. For instance, ReVirt [11] and VMware Workstation [39] implement deterministic replay in a virtual machine monitor. In this case, the unit of record and replay is a virtual machine. The non-deterministic events are the timing and values of interrupts, as well as input received from external devices such as the terminal and network. Alternately, one can implement record and replay in the operating system, as demonstrated by systems such as Flashback [33] and Speck [26]. In that case, the unit of record and replay is a process or group of processes. The logged non-deterministic events are the timing and value of signals, as well as the result of system calls. Record and replay has also been implemented in hardware [25] and at the language level [6]. On multiprocessor systems, the interleaving of shared memory operations is also a source of non-determinism — several systems [1, 12, 23, 28, 37] support multiprocessor record and replay through a combination of logging and offline search.

Researchers have proposed many uses for deterministic replay, primarily targeting desktop and server systems. Deterministic replay has been used to recreate faulty executions to help developers debug code [28, 33]. It has also been used to analyze [22] and remove [21] the effect of intrusions, enhance security by hiding the latency of security checks [26], audit executions performed by an untrusted party [16], and run multiple replicas for fault-tolerance [4].

Until very recently [29], no uses of deterministic replay have been targeted at small, mobile devices such as cell phones. This is unfortunate because, as we argue in the next section, there are several mobile computing applications for which this technology could be particularly helpful.

4. USE CASES FOR MOBILE REPLAY

4.1 Improving response time

The mobile computing community has long been interested in using well-provisioned servers to augment the computational capabilities of mobile computers [2, 9, 8, 14, 31, 32, 34]. Due to size, weight, and power constraints, mobile computers such as cell phones cannot match the CPU, memory, storage, and energy resources offered by servers. Thus, if a phone application offloads some portion of the computation it must perform so that the portion is executed on a trusted server, application performance may improve and energy use may be reduced.

Unfortunately, not all computation can benefit from offload. In order to run a computation remotely, the application state used in that computation must first be sent to the remote server, and the results of the computation must be returned to the mobile computer before the application can resume. Network communication consumes both time and energy. Thus, offload engines must carefully balance the expected savings of performing a computation remotely with the cost of communication when deciding whether to offload a particular unit of computation [3, 9, 14, 15]. Experiments with prior systems have demonstrated that offload, even to nearby servers, generally makes sense only for very large computational tasks such as compilation, speech recognition, natural language translation, augmented reality, face recognition, and games with significant computation.

Another alternative is a thin-client design in which the server hosts the entire application and communicates with the mobile phone only to perform I/O. Since I/Os typically require a synchronous network round-trip, interactive performance may suffer using this design. Additionally, if the phone loses network contact with the server, all application state is unavailable. This, this design is well-suited only for highly reliable mobile networks.

Deterministic replay can significantly expand the scope of applications that benefit from remote execution. It can realize the performance benefits of both offloading and thin-client designs by executing application code on both the phone and server, while simultaneously removing much of the synchronous communication. As with code offload and remote execution, the server used by deterministic replay must be trusted not to leak private data. However, the phone may verify the computation performed by the server using its own execution of the application.

The main idea to improve performance is to run two copies of the computation, one on the mobile computer and another on a remote server. Both copies start from the same initial state and receive the same inputs. Deterministic replay guarantees that both executions produce identical output. Thus, as soon as either execution produces an external output (e.g., network data, screen message, etc.), that output can be used immediately. In some cases, the greater resources of the remote server will enable it to produce output first. In other cases, communication costs will dominate, and the mobile computer will produce the output before it receives the result from the remote server.

Compared to code offload, replay does not require a prediction as to whether remote or local execution will be faster. Replay also guarantees that the state needed for each computation will be reproduced before the application begins that computation; thus, there is no need to ship program state or delta revision to the server prior to each computation. Instead, the two computers need only exchange a log of non-deterministic events and their results. Further, this log can be sent asynchronously as events are generated, as opposed to a state delta which typically is generated and sent synchronously. This allows the server to execute ahead

during compute-intensive portions of an application, while the phone can execute ahead during I/O portions. The user sees the lowest latency of the two.

It is useful to consider what types of events would be in such a log. One source of non-determinism is user input. This typically arrives at a low rate, and so requires little log space. Another source of non-determinism is network input. If the remote server is placed along the network path that connects the mobile computer to the Internet [2], the server can act as a Internet proxy for the mobile computer's network communication with minimal overhead. Since network communication is seen by both parties in this scenario, the log of non-deterministic events can omit the actual data and just include a reference to the observed communication segment. For mobile computers that use more than one network simultaneously [17], it may be best to run the server in the cloud. Non-determinism from file system inputs can be eliminated by replicating file system data on the remote server. A simple way to do this is to run a distributed file system [15] or use a synchronization protocol. A few sources of non-deterministic input will be challenging to handle; for instance, a video camera may produce data at a high rate. For applications that use such data, this form of redundant execution may not make sense.

Other sources of non-determinism are scheduling decisions and signal/interrupt delivery. When the application uses a single processor, these events occur at low frequency, and thus consume little log space. Whenever possible, we plan to use a deterministic scheduling algorithm [10] in which both parties independently make identical scheduling decisions to avoid communicating these events entirely.

If the server executes an application faster, it must send output to be displayed on the phone over the network. For most applications, output events will be relatively small. For instance, it has been shown that it is often faster and more energy-efficient to run a Web browser on a server, ship the graphical result of the rendering of Web pages to the mobile computer, and display it there than it is to run the Web browser on the mobile computer [20].

Replay starts by shipping the current state of the application from the mobile computer to the remote server. For a running application, this cost is roughly proportional to the size of its address space. However, the cost can be mitigated, especially for freshly-started applications, by storing executables and dynamic libraries in a distributed storage system and passing these objects to the server by reference rather than by value.

Both the mobile computer and the server execute the application. When the server reaches a point in the execution that requires a non-deterministic input, it obtains the result locally if possible. This is the case for network input (which the server sees before the mobile computer because it acts as a proxy) and possibly for file system input, if a replicated storage solution is used. For other sources of non-determinism such as user input, the server could potentially wait for the mobile computer to provide the result, which it will do after executing the operation as it performs the identical computation.

A better method is to have the server send a query to the mobile computer asking it to perform the operation immediately. In this case, the mobile computer will not re-execute the operation when it reaches the identical point in its own computation. Instead, it will use the result produced earlier for the server. Using this method, the server can execute multiple operations ahead of the mobile execution, achieving interactive performance similar to that of remote execution.

Both executions are guaranteed to produce identical output, so whichever output is produced first should be used. If the server produces network output first, it immediately transmits the output and squashes the identical output produced by the mobile computer. If the mobile computer produces network output first, the server transmits it immediately and squashes its own output. The server sends all other output, such as screen output, to the mobile computer, which displays the first instance of the output that it receives (either from the server or from its own computation) and squashes the other.

Using the above approach, deterministic replay can improve response time by always using the output produced soonest by either local or remote computation. We expect that the most benefit will occur when different executions produce different classes of output faster; for instance, the mobile computer might produce screen output faster and the server might produce network output faster. The server's proximity to the cloud might also enable it to execute complex interactions with database and other servers faster.

It is possible that one execution may lag significantly behind the other. We expect that I/O delays and idle periods with little computation will usually allow the lagging computation to catch up. However, we can also re-synchronize the executions by shipping the application state from the leading execution to the lagging one (in much the same way that code offload systems exchange state).

Without further optimizations, it is likely that replayed execution will increase energy usage by the mobile computer because it must not only perform the original computation, but it must also exchange non-deterministic events with the remote computer. However, we see several possible paths to reduced energy usage.

First, if applications execute faster, the phone may require less energy since it enters power-save modes sooner. Second, the mobile computer can intentionally omit some operations. For instance, it need not transmit network output since the remote server will do that on its behalf. Third, the mobile computer may decide not to execute some portion of a computation and rely on the remote computer to do it on its behalf (similar to a thin-client approach). However, when the mobile computer resumes computation, it must obtain a state delta from the remote computer in order to re-synchronize its execution. In some cases, such as when the application terminates, no state delta need be sent at all. Interestingly, the synchronization requirements in this scenario is the exact opposite of code offload in that the state delta is sent at the end of a remote computation, rather than at the beginning. It is likely that a combination of these factors will allow the phone to lower energy usage compared to a purely local computation.

4.2 Operation shipping

Deterministic replay can also help reduce the cost of data synchronization for replicated storage systems. Systems such as Mimic [5] have observed that it often requires less network communication to ship a description of the operation that transforms a file to a replica than it takes to ship the transformed file itself. When the transformation has been made on a mobile device, shipping the operation in-

stead of the data can save both time and battery energy. If the remote computer, e.g., a file server, has a copy of the file prior to the operation, it simply re-executes the operation to compute the current value of the data. Non-determinism has been the Achilles' heel of such systems, however. If the remote execution environment does not match the mobile environment precisely, the replicas may diverge. Similarly, innocuous operations such as the execution's incorporating the current time of day may also lead to replica divergence.

With deterministic replay, we guarantee that the replicas will not diverge. Further, since non-determinism is captured comprehensively at an abstraction layer such as the system call or Java virtual machine interface, no application modification is required. The mobile computer can even compare the size of the replay log and the size of the output data to determine which is smaller.

This use of deterministic replay is similar to the approach used by Internet Suspend/Resume to re-migrate virtual machine images among repeatedly-used servers [35]. However, for synchronizing individual files, virtual machine replay may likely capture execution at too coarse a granularity — replay of individual applications seems best.

4.3 Offloading of security and reliability tasks

Security tasks such as virus scanning, taint tracking, and intrusion/anomaly detection designed for desktop environment are usually too heavyweight to be executed on resource-constrained mobile devices without significantly degrading performance and draining battery resources. Similar tasks that improve software reliability, such as detection and avoidance of data races and deadlocks, are also quite heavyweight. Given that servers offer substantially greater computing resources than a typical mobile phone, it is attractive to have servers perform such tasks on behalf of phones.

One can divide security and reliability checks into those that operate on the executing state of an application, such as taint tracking and race detection, and those that require only snapshots of execution state, such as virus scanning. For the later type of tasks, one can ship a digest of the state to be checked to a server, perform the check there, and send the result back to the mobile phone [27]. As described in the previous section, deterministic replay can help for these types of tasks by redoing the computation that produced a large data item rather than transferring the item over the network. Additionally, for a task such as on-access virus scanning, which is performed during each read from and write to disk for suspicious files, deterministic redundant execution of a copy of the application on the server can allow the server to start checks before the phone even reaches the access in its copy of the execution.

For tasks such as taint tracking and intrusion detection that are tightly integrated with the execution of a program, it is infeasible to periodically ship each state to be checked to a remote server. Instead, deterministic replay can be used to execute two versions of the code: a lightweight, unchecked version that runs on the mobile phone, and a heavyweight, verified version that runs on the server. The relative slowdown due to security checks can be mitigated by the difference in the processing power of the two platforms, and possibly by using multiple cores top parallelize checks on the server [26]. We next consider some possible tasks that fall into this category.

First, although prior work has shown that with careful design and engineering, data-flow-based taint tracking can be performed on mobile platforms [13], more comprehensive forms of taint analysis such as direct and implicit control flow analysis require too much processing to be performed on small, mobile clients. With deterministic replay, the server can directly carry out such analysis and also offer a mitigation response, e.g., blocking any network output that disseminates sensitive information. Concurrent with the publication of this paper, the Paranoid Android system [29] used deterministic replay to perform taint tracking on a server in the manner that we propose.

A second example is analysis of the application behavior for detecting anomalies and intrusions, typically associated with resource-based attacks or compromises of the mobile client. In such cases, the application under observation may access sensitive sensors, excessively use resources that drain the phone battery, or simply perform questionable accesses of data and or phone resources. Because of its greater processing power, the server can employ more complex models to detect application behavior while still matching the performance of the phone without such detection tools. However, faithful emulation of the resources on the mobile client, for example through online power modeling support, would be necessary to help detect such stealthy attacks as malware that maliciously drains battery energy [19].

A third example task is run-time software reliability checks such as dynamic deadlock detection and avoidance [18, 38]. Such checks impose a run-time performance penalty but can mitigate software errors that lead to crashes and hangs. Like the previous examples, the performance degradation of the checks can be mitigated by running them only on the server. A complication arises if a check running on the server detects a potential software fault, but the phone has already progressed in its execution beyond the check. One solution is to use checkpoint/rollback [30] to recover application state to a point before the failed check — the phone and server could also collaborate to delay releasing output until the application state on which that output depends has been checked.

In summary, deterministic replay can be useful for many heavyweight mobile security or reliability tasks. A remote server can execute the same application code as the mobile phone, but with additional checks added. As discussed previously, the two executions can run asynchronously — no state other than non-deterministic events need be exchanged. Non-critical output can be released as soon as it is produced by either execution. However, security-critical output, e.g., network messages, should be delayed until the preceding execution is validated by the server.

5. MOBILE REPLAY CHALLENGES

We next examine some of the unique design and implementation challenges for providing a deterministic replay service for small, mobile computers such as cell phones.

One must first choose an abstraction layer in which to guarantee deterministic replay. While replay implemented in hardware and in architectural level virtual machines such as VMware are an attractive choice in desktop and server computing, the diversity in mobile phone hardware makes these options less desirable. Mobile phone architectures often differ substantially from those of servers (e.g., few cell phones use x86 processors). When server and mobile architectures differ, one would have to run an architectural sim-

ulator on a remote server or perform deterministic binary-to-binary translation to guarantee that a remote execution is equivalent to an execution performed on a local phone. The performance overhead of the former technique may be prohibitive, and the implementation challenge of the latter is formidable. Operating system level deterministic replay faces similar challenges.

For these reasons, we believe that a language level virtual machine such as a JVM is the appropriate abstraction level to implement deterministic replay for mobile phones. For mobile operating systems such as Android, the virtual machine based abstraction is particularly suitable, as each application has its own instance of the Dalvik VM. At a minimum, the replay system would need to log all external inputs, scheduling events, and the timing of asynchronous events. This will require some modification to the JVM as the scheduler, interrupts, and other non-deterministic events can only be captured at that level.

The replay system would also need to handle JNI calls that execute functionality outside the purview of the JVM. One possible strategy is to create multiple versions of JNI functions that are guaranteed to produce identical, deterministic effects when executed — this approach seems best for commonly executed library functions. Another strategy is to log all effects of the execution of the JNI call and recreate those effects during replay in lieu of actually re-executing the call. This strategy is less desirable, e.g., it would mean that additional security checks cannot be performed during the JNI code execution. However, this strategy does provide generality in that it can handle even application-specific JNI functions. A final strategy would be to run JNI calls in an architectural simulator — this may be acceptable if such calls comprise a small portion of the application's total execution or if specific use cases, e.g., security checks, require comprehensive coverage.

Another challenge for deterministic replay on mobile phones is resource scarcity. Compute, storage, and battery energy are all precious. Fortunately, deterministic replay has been shown to have overhead of only a few percent [11] for single processor execution. While multiprocessor record and replay can be considerably more expensive, current mobile phones typically do not yet have multiple cores. Even if multicore phone processors become common, one can still restrict each recorded computation to a single core. Storage resources can be conserved by shipping replay logs asynchronously to the remote compute site as they are generated. Further, as argued in Section 4.1, the size of the replay logs can be substantially reduced by removing network input through the use of proxied remote connections.

6. CONCLUSION AND FUTURE WORK

This paper has argued that deterministic record and replay should be a foundational technology for building software systems targeted at mobile phones. While many of the current uses of deterministic replay being explored in the context of desktop and server computing could apply equally well to mobile environments, three potential use cases are especially beneficial when applied to mobile phones: dual execution on cloud or cloudlet computers to reduce latency and possibly phone energy use, offloading of heavyweight security checks to remote servers, and operation shipping for file synchronization. To address the unique challenges presented by mobile phones, we are currently implementing deterministic replay in the Dalvik VMM with a specific focus on supporting uniprocessor replay to reduce resource consumption.

Acknowledgments

We thank the anonymous reviewers for comments that improved this paper. The work is supported by the National Science Foundation under award CNS-0905149. The views and conclusions contained in this document are those of the authors and should not be interpreted as representing the official policies, either expressed or implied, of NSF, the University of Michigan, or the U.S. government.

7. REFERENCES

[1] Gautam Altekar and Ion Stoica. ODR: Output-deterministic replay for multicore debugging. In *Proceedings of the 22nd ACM Symposium on Operating Systems Principles*, pages 193–206, October 2009.

[2] Rajesh Balan, Jason Flinn, M. Satyanarayanan, Shafeeq Sinnamohideen, and Hen-I Yang. The case for cyber foraging. In *the 10th ACM SIGOPS European Workshop*, Saint-Emilion, France, September 2002.

[3] Rajesh K. Balan, M. Satyanarayanan, S. Park, and T. Okoshi. Tactics-based remote execution for mobile computing. In *Proceedings of the 1st International Conference on Mobile Systems, Applications and Services*, pages 273–286, San Francisco, CA, May 2003.

[4] Thomas C. Bressoud and Fred B. Schneider. Hypervisor-based fault tolerance. *ACM Transactions on Computer Systems*, 14(1):80–107, February 1996.

[5] Tae-Young Chang, Araving Velayutham, and Raghpathy Sivakumar. Mimic: Raw activity shipping for file synchronization in mobile file systems. In *Proceedings of the 2nd International Conference on Mobile Systems, Applications and Services*, pages 165–176, Boston, MA, June 2004.

[6] Jong-Deok Choi and Hanri Srinivasan. Deterministic replay of Java multithreaded applications. In *Proceedings of the SIGMETICS Symposium on Parallel and Distributed Tools*, Welches, OR, 1998.

[7] Jim Chow, Tal Garfinkel, and Peter M. Chen. Decoupling dynamic program analysis from execution in virtual environments. In *Proceedings of the 2008 USENIX Technical Conference*, pages 1–14, June 2008.

[8] Byung-Gon Chun and Petros Maniatis. Augmented smartphone applications through clone cloud execution. In *Proceedings of the 8th Workshop on Hot Topics in Operating Systems (HotOS)*, 2009.

[9] Eduardo Cuervo, Aruna Balasubramanian, Dae ki Cho, Alec Wolman, Stefan Saroiu, Ranveer Chandra, and Paramvir Bahl. MAUI: Making smartphones last longer with code offload. In *Proceedings of the 8th International Conference on Mobile Systems, Applications and Services*, pages 49–62, San Francisco, CA, June 2010.

[10] Joseph Devietti, Brandon Lucia, Luis Ceze, and Mark Oskin. DMP: Deterministic shared memory multiprocessing. In *Proceedings of the 2009 International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS)*, pages 85–96, March 2009.

[11] George W. Dunlap, Samuel T. King, Sukru Cinar, Murtaza A. Basrai, and Peter M. Chen. ReVirt: Enabling intrusion analysis through virtual-machine logging and replay. In *Proceedings of the 5th Symposium on Operating Systems Design and Implementation*, pages 211–224, Boston, MA, December 2002.

[12] George W. Dunlap, Dominic G. Lucchetti, Michael Fetterman, and Peter M. Chen. Execution replay on multiprocessor virtual machines. In *Proceedings of the 2008 ACM SIGPLAN/SIGOPS International Conference on*

Virtual Execution Environments (VEE), pages 121–130, March 2008.

[13] William Enck, Peter Gilbert, Byung gon Chun, Landon P. Cox, Jaeyeon Jung, Patrick McDaniel, and Anmol N. Sheth. TaintDroid: An information-flow tracking system for realtime privacy monitoring on smartphones. In *Proceedings of the 9th Symposium on Operating Systems Design and Implementation*, Vancouver, BC, October 2010.

[14] Jason Flinn, Dushyanth Narayanan, and M. Satyanarayanan. Self-tuned remote execution for pervasive computing. In *Proceedings of the 8th Workshop on Hot Topics in Operating Systems (HotOS-VIII)*, pages 61–66, Schloss Elmau, Germany, May 2001.

[15] Jason Flinn, SoYoung Park, and Mahadev Satyanarayanan. Balancing Performance, Energy, and Quality in Pervasive Computing. In *Proceedings of the 22nd International Conference on. Distributed Computing Systems*, Vienna, Austria, July 2002.

[16] Andreas Haeberlen, Paarijaat Aditya, Rodrigo Rodrigues, and Peter Druschel. Accountable virtual machines. In *Proceedings of the 9th Symposium on Operating Systems Design and Implementation*, Vancouver, BC, October 2010.

[17] Brett D. Higgins, Azarias Reda, Timur Alperovich, Jason Flinn, Thomas J. Giuli, Brian Noble, and David Watson. Intentional networking: Opportunistic exploitation of mobile network diversity. In *Proceedings of the 16th International Conference on Mobile Computing and Networking*, Chicago, IL, September 2010.

[18] Horatiu Jula, Daniel Tralamazza, Cristian Zamfir, and George Candea. Deadlock immunity: Enabling systems to defend against deadlocks. In *Proceedings of the 8th Symposium on Operating Systems Design and Implementation*, pages 294–308, San Diego, CA, December 2008.

[19] Hahnsang Kim, Joshua Smith, and Kang G. Shin. On detecting energy-greedy anomalies. In *Proceedings of the 6th International Conference on Mobile Systems, Applications and Services*, Breckenridge, CO, June 2008.

[20] Joeng Kim, Ricardo Baratto, and Jason Nieh. pTHINC: A thin-client architecture for mobile wireless web. In *Proceedings of the 15th International World Wide Web Conference (WWW 2006)*, 2006.

[21] Taesoo Kim, Xi Wang, Nickolai Zeldovich, and M. Frans Kaashoek. Intrusion recovery using selective re-execution. In *Proceedings of the 9th Symposium on Operating Systems Design and Implementation*, Vancouver, BC, October 2010.

[22] Samuel T. King, George W. Dunlap, and Peter M. Chen. Debugging operating systems with time-traveling virtual machines. In *Proceedings of the 2005 USENIX Technical Conference*, pages 1–15, April 2005.

[23] Dongyoon Lee, Benjamin Wester, Kaushik Veeraraghavan, Peter M. Chen, Jason Flinn, and Satish Narayanasamy. Respec: Efficient online multiprocessor replay via speculation and external determinism. In *Proceedings of the 15th International Conference on Architectural Support for Programming Languages and Operating Systems*, pages 77–89, Pittsburgh, PA, March 2010.

[24] Satish Narayanasamy, Cristiano Pereira, Harish Patil, Robert Cohn, and Brad Calder. Automatic logging of operating system effects to guide application-level architecture simulation. In *International Conference on Measurements and Modeling of Computer Systems (SIGMETRICS)*, pages 216–227, June 2006.

[25] Satish Narayanasamy, Gilles Pokam, and Brad Calder. BugNet: Continuously recording program execution for deterministic replay debugging. In *Proceedings of the 32nd Annual International Symposium on Computer Architecture (ISCA)*, pages 284–295, June 2005.

[26] Edmund B. Nightingale, Daniel Peek, Peter M. Chen, and Jason Flinn. Parallelizing security checks on commodity hardware. In *Proceedings of the 13th International Conference on Architectural Support for Programming Languages and Operating Systems*, pages 308–318, Seattle, WA, March 2008.

[27] Jon Oberheide, Kaushik Veeraraghavan, Evan Cooke, Jason Flinn, and Farnam Jahanian. Virtualized in-cloud security services for mobile devices. In *Workshop on Virtualization in Mobile Computing (MobiVirt)*, Breckenridge, CO, June 2008.

[28] Soyeon Park, YuanYuan Zhou, Weiwei Xiong, Zuoning Yin, Rini Kaushik, Kyu H. Lee, and Shan Lu. PRES: Probabilistic replay with execution sketching on multiprocessors. In *Proceedings of the 22nd ACM Symposium on Operating Systems Principles*, pages 177–191, October 2009.

[29] Georgios Portokalidis, Philip Homburg, Kostas Anagnostakis, and Herbert Bos. Paranoid android: Versatile protection for smartphones. In *Proceedings of the Annual Computer Security Applications Conference*, December 2010.

[30] Feng Qin, Joseph Tucek, Jagadeesan Sundaresan, and Yuanyuan Zhou. Rx: Treating bugs as allergies—a safe method to survive software failures. In *Proceedings of the 20th ACM Symposium on Operating Systems Principles*, pages 235–248, Brighton, United Kingdom, October 2005.

[31] Alexey Rudenko, Peter Reiher, Gerald J. Popek, and Geoffrey H. Kuenning. Saving portable computer battery power through remote process execution. *Mobile Computing and Communications Review*, 2(1):19–26, January 1998.

[32] Mahadev Satyanarayanan, Paramvir Bahl, Ramon Caceres, and Nigel Davies. The case for VM-based cloudlets in mobile comptuing. *IEEE Pervasive Computing*, 8(4):14–23, October–December 2009.

[33] Sudarshan Srinivasan, Christopher Andrews, Srikanth Kandula, and Yuanyuan Zhou. Flashback: A light-weight extension for rollback and deterministic replay for software debugging. In *Proceedings of the 2004 USENIX Technical Conference*, pages 29–44, Boston, MA, June 2004.

[34] Ya-Yunn Su and Jason Flinn. Slingshot: Deploying stateful services in wireless hotspots. In *Proceedings of the 3rd International Conference on Mobile Systems, Applications and Services*, pages 79–92, Seattle, WA, June 2005.

[35] Ajay Surie, H. Andres Lagar-Cavilla, Eyal de Lara, and M. Satyanarayanan. Low-bandwidth VM migration via opportunistic replay. In *Proceedings of the 9th Workshop on Mobile Computing Systems and Applications (HotMobile)*, Napa, CA, February 2008.

[36] Joseph Tucek, Shan Lu, Chengdu Huang, Spiros Xanthos, and Yuanyuan Zhou. Triage: Diagnosing production run failures at the user's site. In *Proceedings of the 21st ACM Symposium on Operating Systems Principles*, pages 131–144, October 2007.

[37] Kaushik Veeraraghavan, Dongyoon Lee, Benjamin Wester, Jessica Ouyang, Peter M. Chen, Jason Flinn, and Satish Narayanasamy. Doubleplay: Parallelizing sequential logging and replay. In *Proceedings of the 16th International Conference on Architectural Support for Programming Languages and Operating Systems*, Newport Beach, CA, March 2011.

[38] Yin Wang, Terence Kelly, Manjunath Kudlur, Stephane Lafortune, and Scott Mahlke. Gadara: Dynamic deadlock avoidance for multithreaded programs. In *Proceedings of the 8th Symposium on Operating Systems Design and Implementation*, pages 281–294, San Diego, CA, December 2008.

[39] Min Xu, Vyacheslav Malyugin, Jeff Sheldon, Ganesh Venkitachalam, and Boris Weissman. ReTrace: Collecting execution trace with virtual machine deterministic replay. In *Proceedings of the 2007 Workshop on Modeling, Benchmarking and Simulation (MoBS)*, June 2007.