Low-Bandwidth VM Migration via Opportunistic Replay

Ajay Surie
Carnegie Mellon University

H. Andrés Lagar-Cavilla
University of Toronto

Eyal de Lara
University of Toronto

M. Satyanarayanan
Carnegie Mellon University

ABSTRACT

Virtual machine (VM) migration has been proposed as a building block for mobile computing. An important challenge for VM migration is to optimize the transfer of large amounts of disk and memory state. We propose a solution based on the *opportunistic replay* of user interactions with applications at the GUI level. Whereas this approach results in very small replay logs that economize network utilization, replay of user interactions on a VM at the migration target site can result in divergent VM state. Cryptographic hashing techniques are used to identify and transmit only the differences. We discuss the implementation challenges of this approach, and present encouraging results from an early prototype that show savings of up to 80.5% of bytes transferred.

Keywords

Virtual Machines, VM migration, VM Replay, Internet Suspend/ Resume®

1. INTRODUCTION

The use of virtual machine (VM) technology in mobile computing has attracted significant attention in the last few years [3, 4, 6, 7]. Whereas there are differences in the specifics of their implementations, in all these projects the user's computational environment is migrated between hosts by suspending the VM on the source host, transferring its state, and resuming it on the destination. VM migration simplifies many aspects of moving computation from one site to another by encapsulating an entire execution state with no need for kernel, library or language support at the migration target. However, this simplification comes at a price: a VM is typically large, often tens of GB in size. At the low bandwidths that are common in mobile computing, the transfer time for such a large amount of VM state is excessive.

In this paper we introduce *opportunistic replay*, a technique that optimizes the migration of VMs between frequently visited hosts. Opportunistic replay builds on the key observation that replay does not have to be perfect to be useful. A complete log of user interactions with a VM, such as keystrokes and mouse movements, is first captured. The log is then shipped to a remote site, and replayed there on an identically-configured VM in the same initial state. Under ideal conditions, one would intuitively expect these steps to produce identical final VM state at the remote site. However, as explained in Section 2, conditions are rarely ideal. Consequently, some replays may produce final VM state that is close, but not identical, to the desired final state. We use cryptographic hashes to detect overlapping VM state at source and destination, and only ship the missing pieces. If the log is relatively small and if replay is close to ideal, this approach realizes significant reductions in total network transmission. However, there is never any loss of correctness even in the worst case when replay produces no overlapping state. The only negative impact in that case is performance degradation due to the overhead of log capture, shipping and replay.

In this work we present a study on the feasibility of opportunistic replay. We explore the challenges and implementation issues that a complete replay system will face. We present an experimental prototype that addresses an initial subset of the challenges identified. We study its performance when migrating a VM over different domestic-class bandwidths and after performing four different types of representative end-user workloads. Our prototype is able to realize savings of up to 80.5% in terms of bytes transferred.

2. CHALLENGES OF VM REPLAY

Many factors complicate the conceptual simplicity of opportunistic replay. These can be grouped into three broad categories that we discuss below: (i) incomplete log capture; (ii) non-deterministic externalities; and (iii) exactly-once size effects.

Incomplete log capture:. An obvious prerequisite for ideal replay is complete capture of all external stimuli that could perturb VM state. This includes external interrupts and data transfers from storage devices and networks, as well as user input via keyboard, mouse and other human interaction devices. A potential concern is the size of the log necessary for complete capture. In their work on VM logging and replay for intrusion analysis [5], Dunlap et al. report log growth ranging from 0.04GB per day to 1.2GB per day. Further, VMware report for their ReTrace tool a compressed log size of roughly 776 KB, without accounting for network activity, when rebooting a Windows XP VM [16]. Both sources report only modest CPU overhead for complete logging. However, logs growing at these rates would be impractical for VM relocation, since they can easily surpass the size of the actual state changes that would be shipped with standard VM migration techniques. An additional concern pertains to inserting logging code in closed-source

VMMs such as VMware Workstation, and closed-source guest operating systems such as Windows XP. Without access to the source code of these components, it may be impossible to ensure complete log capture. Just capturing user interaction is simpler, since the windowing system provides a natural interface for interposition of logging code. This will produce short logs that, even though incomplete, can still be leveraged by opportunistic replay to realize significant reductions in VM state transmission.

Non-deterministic externalities:. Deterministic code execution is another obvious requirement for replay to produce VM state identical to the original execution. A major source of non-determinism in interactive systems is network access to Web sites with dynamically generated content. Consider, for example, Web access to a site that offers current stock quotes. Replaying the Web access may result in different content being returned because stock prices have changed. Even when the intrinsic content of a Web page is unchanged, there may be parasitic content such as advertising that is different on replay. While there is no "solution" per se to the problem of non-deterministic externalities, opportunistic replay is able to cope because replay does not have to be perfect to be useful. Only when there is an excessive amount of non-determinism will the overlap between local and remote VM state drop below a useful level.

Exactly-once side effects:. A difficult problem for VM replay is the occurrence of events that should not be replayed for reasons of correctness. Consider, for example, an interactive session in which a user sends an email message. Replaying this action would result in duplicate message transmission, which clearly violates correctness. A safe solution is to block any outbound network traffic during replay; this will result in divergent and less beneficial replay, but will guarantee consistency with the outside world. A solution with better replay performance would detect logged actions that have such exactly-once side effects and skip them during replay. Although this is a very difficult problem in its most general form, it may be relatively simple to perform conservative detection of common cases. For example, if there are a set of known Web sites at which a particular user performs financial transactions, the log can be examined for operations that reference these Web sites. An even more conservative approach is to suppress replay of all secure Web operations, on the grounds that high-value transactions are almost certain to occur only within the scope of a secure Web session. Finally, one could analyze network traffic and permit "read-only" transactions on well-known protocols, such as POP3's `STAT`, `LIST` and `RETR`, but not `DELE`.

3. PROTOTYPE IMPLEMENTATION

We have implemented a prototype to explore the feasibility of opportunistic replay. Our prototype uses Xen [2] version 3.1. In order to efficiently detect modified disk blocks, we rely on a custom block device driver that exports a virtual disk to a VM. The block device is coupled with a user space daemon that chunks the data written to it and keeps track of dirty disk chunks. To efficiently synchronize VM memory images, we use the `rsync` utility [15]. For record and replay operations, we leverage Xnee [11], an off-the-shelf tool for the X11 environment. Xnee synchronizes replay against the X11 windowing events registered during record (e.g. change of focus, window pop-up, etc). Finally, our prototype does not yet address the problem of replaying interactions with exactly-once side effects.

We refer to the *source* as the host at which a user is currently interacting with a VM, and *destination* as the host to which the VM will eventually be migrated. We assume that both source and destination hosts have initially identical copies of a suspended VM, which is the typical setup in VM migration-based mobile computing prototypes like the Internet Suspend/Resume system® (ISR) [6, 12]. Our implementation involves 3 steps:

- **Capture:** All keyboard and mouse input to the locally running VM is recorded to a log. During recording, our virtual block device tags dirty disk state, and keeps track of the order in which disk chunks are modified. This will make synchronization more efficient in later stages.
- **Replay:** The recorded interaction log is transferred to and replayed on the VM at the destination. Replay speed may be faster than capture speed. This suppresses think time during replay and takes advantage of any free compute resources at the destination. Modified disk chunks at the source are shipped to the destination in the background while replay is in progress. This ensures that network bandwidth is not wasted by being idle during replay. In environments where network bandwidth may be scarce or costly, in monetary or power consumption terms, we can disable this optimization. Since replay may be imperfect, state produced later in the replay may diverge more than state produced at the beginning. To optimize for this tendency, dirty disk chunks are shipped to the destination in reverse of the order that they were modified at the source. New VM disk state generated by replay at the destination is also tagged for synchronization.
- **Synchronization:** After replay, any residual state at the source must be transferred to the destination so that the resulting state at both hosts is identical. Under ideal conditions, the amount of overlapping state generated at the source and destination is large, so there is little residual state to be transferred. Differences between the state at the source and destination are detected via cryptographic hashing using the SHA-1 algorithm, and propagated to the destination.

4. RESULTS

We conducted experiments using different workloads at various emulated bandwidths. We ran each workload at the source host on a VM in the same initial state and saved the resulting VM state and interaction log. We then performed opportunistic replay of the saved log at the destination, followed by transfer of residual disk and memory state from the source. For comparison, we also transferred all modified memory and disk state to the destination without replay. We enabled background transfer of state during replay, which biases results against our prototype for environments with limited or costly connectivity. All results we present below are an average of 3 runs.

Table 1 describes the workloads we used in our experiments. General simulates a user conducting everyday tasks such as reading news websites (e.g, `CNN.com`) with Mozilla Firefox, and editing documents with the OpenOffice.org software suite and the Gedit text editor. Install installs applications such as the Emacs text editor and the Mozilla Thunderbird email client using Synaptic, a graphical package managing tool available for the Ubuntu distribution. Kbuild downloads the Linux kernel version 2.6.23.1 from `www.kernel.org` and builds it without modules enabled. Finally, the Gimp program is used in the last workload to edit and

Workload	Description	Duration (minutes)	Log Size [Gzipped] (MB)	Dirty Disk State (MB)	Dirty Memory State (MB)
General	Web browsing Document editing	15.65	0.66 [0.15]	11.57	117.43
Install	Application installation	2.85	0.11 [0.02]	57.08	105.18
Kbuild	Kernel download & build	7.23	0.20 [0.04]	153.51	144.47
Gimp	Image manipulation	9.73	0.81 [0.18]	6.73	48.93

Table 1: Interactive workloads

Bandwidth	Upstream (Mbits/sec)	Downstream (Mbits/sec)
Cable	0.375	6
DSL	0.75	3
EVDO Rev A	1.2	3.8

Table 2: Emulated bandwidths in Mbits/sec (upstream/downstream)

manipulate sample images. As shown in the right hand columns of Table 1, the size of the captured logs is three orders of magnitude smaller than the total amount of dirty state generated by the respective workload.

In our experimental configuration, the source host was configured with an Intel Pentium 4 3.60GHz CPU, 2MB cache and 2GB of RAM, and the destination host was configured with a 2.66GHz Intel Core 2 quad-core CPU, 4MB cache, and 4GB RAM. We used Ubuntu 7.04 as the guest and host OS on both machines. The VM was configured with 512MB of RAM and a 4GB virtual disk. Experiments were conducted using emulated bandwidths, shown in Table 2. The Cable and DSL bandwidth values were selected from the current offerings of two large ISPs. The EVDO Revision A standard is currently in use by wireless network providers such as Verizon and Sprint. We note that by emulating advertised peak bandwidths we bias against the benefits of opportunistic replay.

Replay at native speed: . Our first set of experiments demonstrates the benefits of replay when the log is replayed at native (capture) speed. Figure 1 (a) and Figure 1(b) show the amount of memory and disk state transferred during migration with and without replay at the destination at various bandwidths. The figure shows that the wins are greatest for the Kbuild workload, where disk and memory state transferred at Cable speed shrink from 153.51MB to 29.74MB, and from 144.47MB to 28.9MB, respectively. As described in Section 3, dirty disk state is shipped in the background while replay is in progress. A higher upstream bandwidth results in more disk state shipped in the background and not being generated through replay. For example, at EVDO speeds, disk state transfer for the Install workload is reduced from 57.08MB to 44.73MB. In contrast, at cable speeds, disk state transfer is reduced to 14.06MB. If the amount of modified state is small or bandwidth is high, the background disk state transfer might finish before replay is complete, and replay is terminated; this holds true for the General and Gimp workloads. Early replay termination also limits the amount of memory state being recreated. For example, with the General workload, 107.55MB of memory state is shipped at EVDO speeds, which is reduced to 58.63MB at Cable speeds. Thus, even for interactive workloads that generate little dirty disk state, significant memory state shipping savings can be realized.

Figure 2 (a) and (b) show the reduction in migration time using replay at native speed for both disk and memory state. Replay significantly reduces disk state transfer times, which is most apparent at the lowest bandwidths. At Cable speed, transfer time for disk state reduces from 1373 seconds to 342 seconds for the Install workload, and from 3615.6 seconds to 767 seconds, for Kbuild. This trend is also true of memory state transfer, with the Install workload obtaining the largest reduction, at Cable speed. For Install, transfer times are reduced from 2459.8 seconds to 400 seconds. For the General & Gimp workloads, transfer times at Cable speed go down from 2744.5 to 1365.5 seconds, and from 1137.2 to 894.5 seconds, respectively.

Replay at speeds higher than native:. The goal of higher speed replay is to suppress idle or think time during replay, in order to generate more VM state in less time at the destination. Figures 3 (a) and (b) show the benefits of higher speed replay for General and Gimp, our most interactive workloads. In the figures, 2X replay indicates that think time between interactions was compressed by half during replay. We show results only for EVDO speeds; by compressing think time, more of the log is replayed before the background transfer of disk state finishes. This reduces the amount of memory state transferred. The results at other bandwidths do not show improvements as the entire log is replayed even at native speed. Kbuild and Install also show little improvement, since they are CPU intensive workloads that expose almost no compressible think times. Finally, the amount of dirty disk state for Gimp and General is small and not significantly affected by faster replay.

The General workload benefits from faster replay, with memory state transfer decreasing from 117.43MB with no replay to 59.64MB when the session is replayed at four times the native speed. The same trends translate to Figure 3 (b), with transfer time decreasing from 576.17 seconds to 299.17 seconds. The Gimp workload does not significantly benefit from faster replay as it does not generate much dirty memory state.

5. RELATED WORK

Closest in spirit to opportunistic replay is the work of Lee et al. on operation shipping for mobile file systems [8]. That work showed how logging and replay could significantly improve performance in propagating large files from a weakly-connected client to a server. Our work differs in two major ways. First, our focus is on recreating VM state rather than file system state. Second, we use an

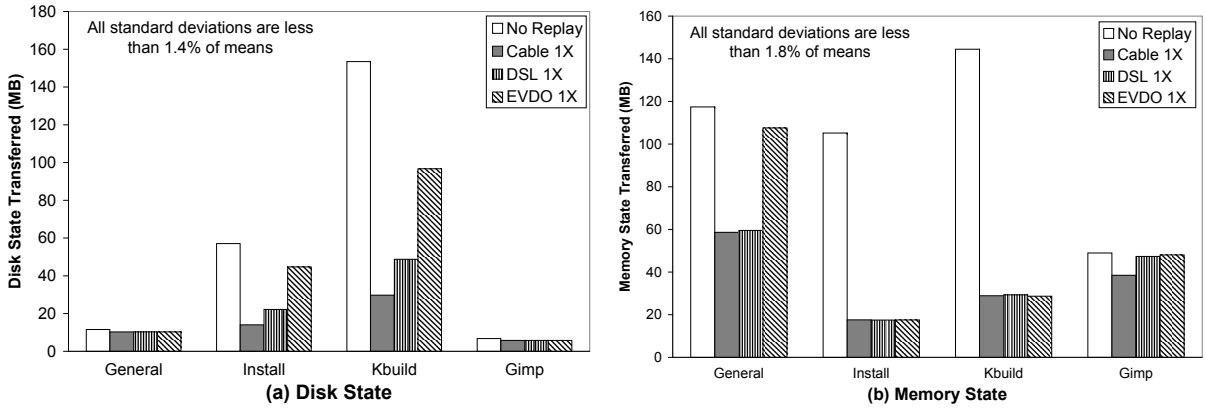

Figure 1: Bytes transferred during replay at native speed

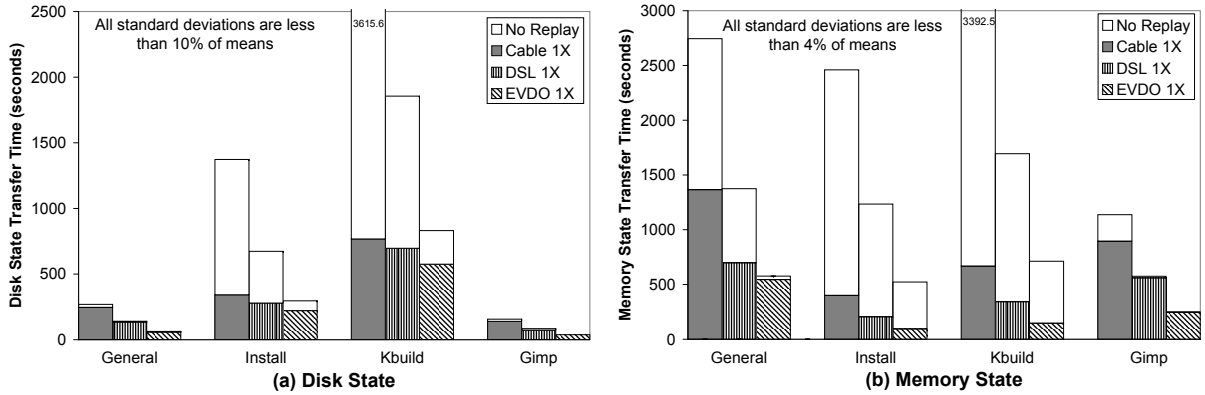

Figure 2: Total state transfer time at native speed. ("Replay" bars superimposed on top of "No Replay" bars)

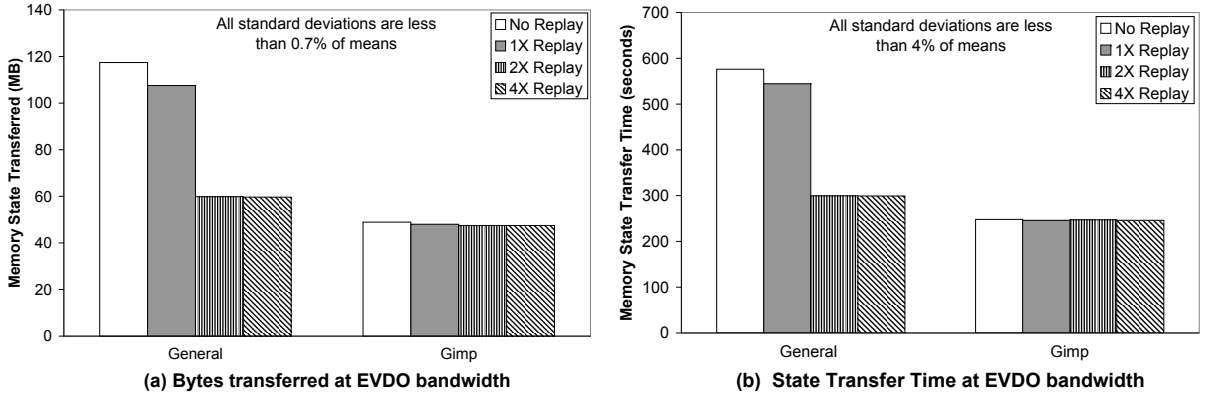

Figure 3: Bytes transferred and transfer time for memory state using higher speed replay

opportunistic approach and can therefore benefit even from replays that diverge from the original execution.

Our use of an opportunistic approach to exploiting data similarity was inspired by the work of Tolia et al. in distributed file systems [13] and relational databases [14]. A recurring theme in that work is the description of a large data object in recipe form using cryptographic hashes, and the synthesis of parts of that object from local data sources in order to reduce transmissions over a bandwidth-challenged network. More recently, Annapureddy et al. have used similar techniques in the context of cooperative caching for file servers [1]. Our work extends the opportunism underlying these approaches to the realm of VMs, using techniques specific to log capture and replay. At an even deeper level, the idea of using cryptographic hashing for detecting similarity of data content has been used in file systems such as LBFS [10], and the `rsync` file transfer protocol [15].

As mentioned in Section 2, the concept of VM logging and replay was introduced by Dunlap et al. [5] in the context of the ReVirt system for intrusion analysis. Since their work was dependent on complete log capture and ideal replay, it required complex source code modifications to the VMM, and generated very large logs. In contrast, opportunistic replay can afford to be less strict. As we have shown in this paper, a profitable level of fidelity of log capture and replay can be implemented without VMM modifications.

6. CONCLUSION AND FUTURE WORK

We have presented opportunistic replay, a technique that replays user actions on a target site to recreate VM state and minimize the overhead of VM migration. By acknowledging that replay for VM migration does not need to be perfect to be useful, we are able to attack the problem at the GUI level, greatly reducing complexity and producing brief logs, while still yielding byte transfer reductions of up to 80.5%.

While our current prototype presents encouraging evidence, a number of implementation aspects remain unexplored. First, tracking VM memory state as it is dirtied will further optimize our network transfer savings. In addition, since we tolerate imperfect replay, the likelihood that final VM state diverges from state produced by replay increases with the length of the interaction log. To tackle this problem, we plan to explore *incremental replay*, in which a single user work session is divided into a number of segments of shorter duration. Each segment has its own interaction log and VM state at the destination is synchronized with that at the source when replay for a segment is complete. As a result, state at the destination only lags slightly behind that at the source. This bears resemblance to the concept of trickle reintegration in the Coda file system [9]. Finally, policies to prevent replay from altering outside-world state (such as email inboxes or bank accounts) need to be developed. Based on the promising results reported here, we also plan to integrate opportunistic replay into ISR.

Acknowledgements

We thank Niraj Tolia for his involvement in the initial stages of this work. We also thank Mike Kozuch and the anonymous reviewers for their encouraging comments and suggestions. This research was supported by the National Science Foundation (NSF) under grant number CNS-0509004, and the National Science and Engineering Research Council (NSERC) of Canada under grant number 261545-3 and a Canada Graduate Scholarship. Any opinions, findings, conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of the NSF, NSERC, Carnegie Mellon University, or the University of Toronto. Internet Suspend/Resume is a registered trademark of Carnegie Mellon University. All unidentified trademarks mentioned in the paper are properties of their respective owners.

7. REFERENCES

[1] S. Annapureddy, M. J. Freedman, and D. MaziÃĺres. Shark: Scaling file servers via cooperative caching. In *Proc. 2nd Symposium on Networked Systems Design and Implementation*, Boston, MA, May 2005.

[2] P. Barham, B. Dragovic, K. Fraser, S. Hand, T. Harris, A. Ho, R. Neugebauer, I. Pratt, and A. Warfield. Xen and the Art of Virtualization. In *Proc. 19th Symposium on Operating Systems Principles (SOSP)*, Oct. 2003.

[3] Caceres, R., Carter, C., Narayanaswami, C., Raghunath, M. Reincarnating PCs with Portable SoulPads. In *Proc. 3rd International Conference on Mobile Systems, Applications and Services (MobiSys)*, Seattle, WA, June 2005.

[4] L. Cox and P. Chen. Pocket hypervisors: Opportunities and challenges. In *Proc. HotMobile 2007: the 8th Workshop on Mobile Computing Systems and Applications*, Tucson, AZ, February 2007.

[5] G. W. Dunlap, S. T. King, S. Cinar, M. A. Basrai, and P. M. Chen. ReVirt: Enabling Intrusion Analysis through Virtual-Machine Logging and Replay. In *Proc. 5th Symposium on Operating Systems Design and Implementation*, Dic 2002.

[6] M. Kozuch and M. Satyanarayanan. Internet Suspend/Resume. In *Proc. 4th IEEE Workshop on Mobile Computing Systems and Applications*, Callicoon, NY, June 2002.

[7] H. A. Lagar-Cavilla, N. Tolia, E. de Lara, M. Satyanarayanan, and D. O'Hallaron. Interactive Resource-Intensive Applications Made Easy. In *Proc. Middleware 2007: ACM/IFIP/USENIX 8th International Middleware Conference*, Newport Beach, CA, November 2007.

[8] Y.-W. Lee, K.-S. Leung, and M. Satyanarayanan. Operation Shipping for Mobile File Systems. *IEEE Transactions on Computers*, 51(12), 2002.

[9] L. B. Mummert, M. R. Ebling, and M. Satyanarayanan. Exploiting weak connectivity for mobile file access. In *Proc. of the 15th Symposium on Operating Systems Principles (SOSP)*, pages 143–155, New York, NY, USA, 1995. ACM Press.

[10] A. Muthitacharoen, B. Chen, and D. Mazieres. A Low-Bandwidth Network File System. In *Proc. 18th Symposium on Operating Systems Principles (SOSP)*, Chateau Lake Louise, Banff, Canada, Oct. 2001.

[11] H. Sandklef, J.-E. Dahl, and L. Santander. GNU Xnee. `http://http://www.sandklef.com/xnee/`.

[12] M. Satyanarayanan, B. Gilbert, M. Toups, N. Tolia, A. Surie, D. R. O'Hallaron, A. Wolbach, J. Harkes, A. Perrig, D. J. Farber, M. A. Kozuch, C. J. Helfrich, P. Nath, and H. A. Lagar-Cavilla. Pervasive Personal Computing in an Internet Suspend/Resume System. *IEEE Internet Computing*, 11(2), 2007.

[13] Tolia, N., Kozuch, M., Satyanarayanan, M., Karp, B., Bressoud, T., Perrig, A. Opportunistic Use of Content-Addressable Storage for Distributed File Systems. In *Proc. 2003 USENIX Technical Conference*, San Antonio,

TX, June 2003.

[14] Tolia, N., Satyanarayanan, M., Wolbach, A. Improving Mobile Database Access over Wide-area Networks without Degrading Consistency. In *Proc. 5th International Conference on Mobile Systems, Applications and Services (MobiSys)*, 2007.

[15] A. Tridgell and P. Mackerras. The rsync algorithm. Technical Report TR-CS-96-05, Department of Computer Science, The Australian National University, Canberra, Australia, 1996.

[16] M. Xu, V. Malyugin, J. Sheldon, G. Venkitachalam, and B. Weissman. Retrace: Collecting execution trace with virtual machine deterministic replay. In *Proc. 3rd Annual Workshop on Modeling, Benchmarking and Simulation, MoBS*, San Diego, CA, June 2007.