# Portable Storage Support for Cyber Foraging

*Ya-Yunn Su and Jason Flinn*
*Department of Electrical Engineering and Computer Science*
*University of Michigan*

## Abstract

*The ubiquitous computing community has focused considerable attention on enabling resource-poor mobile computers such as cell-phones and handhelds to execute demanding applications such as speech recognition and virtual desktops. One proposed solution, cyber foraging, uses remote computers located at wireless hotspots to execute application services on behalf of mobile computers. In this paper, we explore how portable storage can improve the performance of a cyber foraging infrastructure. Our approach uses portable storage to store snapshots of service state along with logs that are used for deterministic replay. We show that this approach reduces the time to instantiate new services at wireless hotspots by up to 85% when used with the Slingshot cyber foraging infrastructure.*

## 1 Introduction

Creating applications that execute on small, mobile computers is a challenging task. On one hand, the size and weight constraints of handheld and similar computers limit their processing power, battery capacity, and memory size. On the other hand, users' appetites are driven by the applications that run on their desktops; these often require more resources than a handheld can provide. A solution to this dilemma is remote execution using wireless networks to access compute servers; this combines the mobility of handhelds with the processing power of desktops. However, a mobile user that accesses a distant, well-known compute server from a wireless hotspot often finds that the latency and bandwidth limitations of the backhaul connection between the hotspot and the Internet substantially degrade the performance of interactive applications.

The ubiquitous computing research community has addressed this limitation through the development of *cyber foraging* infrastructure that allows mobile clients to leverage third-party *surrogate* computers located at wireless hotspots [2, 5, 6, 7, 12]. Since these surrogates are accessible via high-bandwidth (54 Mb/s for 802.11g), low-latency wireless networks, interactive response time is much improved.

Unfortunately, in our experience with the Slingshot cyber foraging infrastructure (described in the next section), we have found that the same backhaul limitations that motivate the need for cyber foraging can also substantially delay the instantiation of new services on surrogate computers. While the wireless bandwidth at the hotspot is plentiful, the state needed to instantiate a surrogate must be shipped over a backhaul connection shared among many users. In this paper, we examine how portable storage devices can substantially reduce that delay by shipping this state locally over the wireless network instead.

Specifically, we address the following questions:

- What data should be stored on portable storage to best support cyber foraging?
- How can stateful services such as remote desktops be handled efficiently?
- How much can portable storage reduce the time to instantiate new services?
- What are the storage requirements for some typical mobile services?

We begin in the next section with an overview of Slingshot. Section 3 lays out the design and implementation of our portable storage support, and Section 4 evaluates its effectiveness. We conclude with a discussion of related and future work.

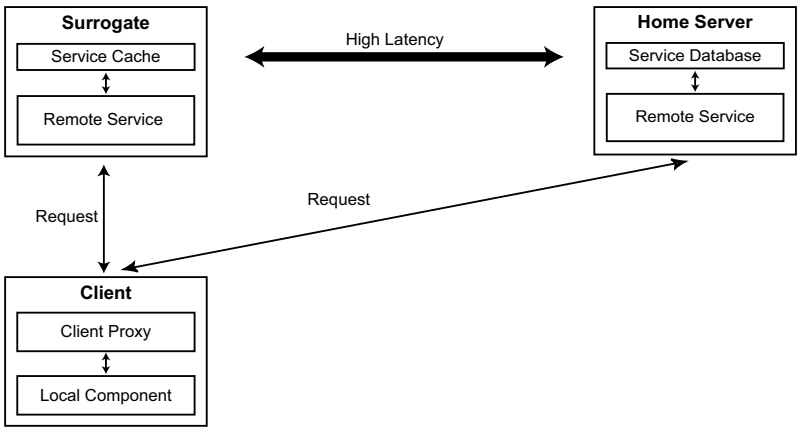



**Figure 1**. Slingshot architecture

## 2 Slingshot Overview

Figure 1 shows an overview of the Slingshot cyber foraging infrastructure [10]. Each Slingshot application is partitioned into two components: a *local component* that runs on the mobile client and a *remote service* that is replicated on the home server and surrogates. Ideally, we partition the application so that resource-intensive functionality executes as part of the remote service—the local component typically contains only the user interface. This partitioning enables demanding applications to be run on mobile computers that are highly portable but also extremely resource-impoverished.

A first-class replica of each remote service executes on the home server and second-class replicas execute on nearby surrogates. The home server is a known, well-maintained server under the administrative control of the user. For instance, the home server might be the user's desktop or a shared server maintained by an IT department. In contrast, surrogates are co-located with wireless access points. They are administered by third parties and are not assumed to be reliable.

Slingshot broadcasts application requests to all replicas and returns the first response it receives to the local application component. Second-class replicas at wireless hotspots improve interactive performance because their responses are received faster than those from the distant home server. The first-class replicas provide a well-maintained repository for application state that persists even if all surrogates fail.

Each replica executes within its own VMware [13] virtual machine. Replica state consists of two components: the *persistent state*, or disk image of the virtual machine, and the *volatile state*, including the memory image and registers of the virtual machine. To handle the persistent state, we use the Fauxide and Vulpes modules developed by Intel Research's Internet Suspend/ Resume project [7]. These modules intercept VMware disk I/O requests. On the home server, we redirect these requests to a *service database* that stores the disk blocks of every remote service. On a surrogate, VMware reads are first directed to a *service cache* — if the block is not found in the cache, it is fetched from the service database on the home server.

When a mobile computer connects to a new hotspot and Slingshot detects a nearby surrogate, it will instantiate a new second-class replica on that surrogate. Slingshot checkpoints the first-class replica on the home server and sends the volatile state associated with the checkpoint to the surrogate. This is a lengthy process, during which the user may continue to execute the application using existing replicas (albeit with potentially poor response time). Slingshot logs all operation activity that occurs during this time period. Once it has retrieved the volatile state and started a second-class replica on the nearby surrogate, it replays these logged operations to bring the state of the replica up-to-date. Slingshot applications provide wrappers that enforce determinism at the application level (similar to Rodrigues' BASE [8]), guaranteeing that replicas remain consistent. As the replica executes fur-

ther operations, blocks associated with its persistent state are fetched on demand and cached.

## 3 Design and implementation

### 3.1 Overview

For a typical service, the volatile state is 145 MB in size and the persistent state is 4 GB. Using compression and Waldspurger's ballooning technique [14], the volatile state can be reduced to 30–40 MB. This takes approximately 30 minutes to transfer from the home server to the surrogate if the home server is connected to the Internet by a 256 Kb/s DSL link.

In this section, we describe how we can substantially reduce the time to instantiate a service on a nearby surrogate by storing its state on a portable storage device such as a mobile disk drive or flash card. Specifically, for each service, we store the following types of state:

- **Volatile state.** After ballooning and compression are applied, the entire remaining volatile state is needed to instantiate a new replica. Typically, the volatile state of two different virtual machines are almost completely different from one another. Therefore, we store the volatile state of each service as a file on the portable storage device.
- **Persistent state.** In contrast to the volatile state, only a relatively small portion of the persistent disk image may actually be read by a replica while it executes on a surrogate. Further, two different services may share many disk blocks in common, especially if they were created from the same base operating system. Given these observations, we use content-addressable storage, as described in Section 3.2 to store the persistent state.
- **Operation log.** This stores all remote operations performed by an application. For a stateful application such as a remote desktop, the operation log can be used by the mobile client to bring a new replica up-to-date with the existing state.

When a user returns to her home server, she creates new checkpoints for her existing Slingshot applications. Each checkpoint consists of a snapshot of the volatile and persistent state of the remote service running on the home server. Because the user is co-located with the home server at this point, the snapshots can be quickly and efficiently uploaded to the portable storage device.

When the user creates a new snapshot of a service, the operation log is empty. When she uses the application on the road, Slingshot appends each application operation to the log. This enables Slingshot to instantiate a new replica of a stateful service by first restoring the checkpoint represented by the volatile state, and then deterministically replaying the operation log. As an optimization, services may be specified as being stateless—in this case, Slingshot neither records nor replays the operation log.

### 3.2 The persistent state

Previous research in virtual machine migration by Sapuntzakis [9] and Tolia [11] has shown that content-addressable storage is highly effective in reducing the storage and transfer costs of persistent state. We adopt their approach by dividing the virtual disk into 4 KB chunks and indexing each chunk by its SHA-1 hash value. As shown in Figure 2, the portable storage stores a chunk table for each service and a chunk database that contains the 4 KB disk blocks. The chunk table maps logical block numbers to the SHA-1 hash of the data stored at each location. A hash table maps the SHA-1 value to the location of the data in the chunk database. Since SHA-1 has been shown to be extremely collision-resistant, the probability of any two blocks with different data hashing to the same value is infinitesimal.

When a new replica is instantiated on a nearby surrogate, the mobile computer tries to find the service state for that replica on any available portable storage device. If the state is found, the volatile state, chunk table, and hash table are transmitted to the surrogate over the hotspot's wireless network. Since the local network typically offers much higher bandwidth than the backhaul connection, retrieving the data from a local portable storage device is much quicker than retrieving it from the home server. One reason that we transmit the chunk table and the hash

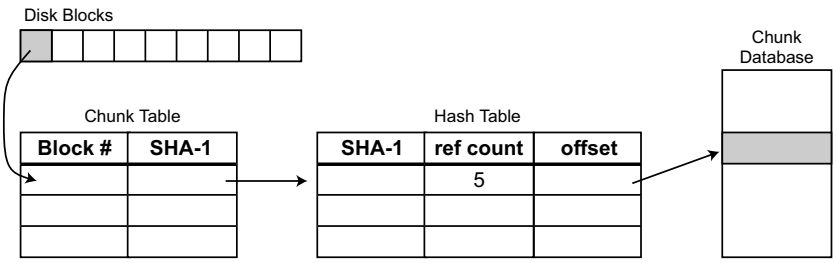



**Figure 2**. Reading persistent state

table to the surrogate is that the surrogate can usually maintain this information in memory, whereas a resource-constrained mobile computer cannot. By placing this information on the surrogate, Slingshot improves the latency of accessing persistent state.

### 3.3 The operation log

The operation log is needed only for stateful services. The result of a remote operation for a stateful service depends upon the operations that it has previously executed. Slingshot ensures that applications are deterministic; i.e. given two replicas in the same initial state, an identical sequence of operations sent to each replica produces identical results.

When a stateful service is instantiated from portable storage, its volatile state will not match the stored state after it executes any remote operation. Further, the execution of an operation will often change the persistent state causing dirty blocks to be written to the service cache on the surrogate. Thus, if a user were to leave a hotspot at which she had executed some operations, and then instantiate a new replica from the state on portable storage at another hotspot, the new replica would not contain any of the modifications she had made at the previous hotspot. This is clearly unacceptable.

One solution to this problem would be to download the volatile state and modifications to the persistent state from a surrogate before leaving a hotspot. Unfortunately, this requires explicit notification that a mobile computer will be leaving its present location. Further, even if such notification is given, sufficient notice must be given so that the state can be downloaded before the movement occurs. This would typically require several minutes of advance notice. When such notice is unavailable, the mobile computer would be required to retrieve the modified state over a limited Internet backhaul connection, leaving a half-hour or more until another replica can be instantiated. Given these considerations, we rejected this alternative as infeasible.

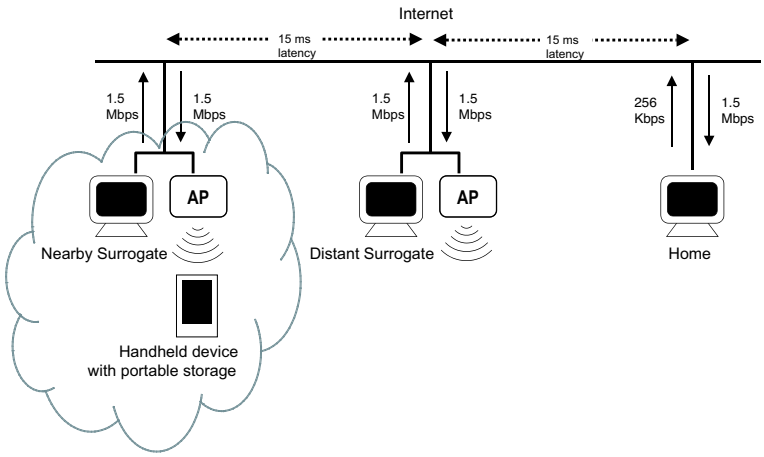
Our approach to supporting stateful services is to store a log of all remote operations on the portable storage device. Since Slingshot ensures determinism, a replica can be brought up-to-date by restoring the checkpoint represented by the volatile and persistent state and then replaying all operations that have occurred since that checkpoint was taken at the home server. When a remote operation is performed while the user is on the road, that operation is appended to the log. Thus, the size of the operation log continues to increase until the user returns home and a new checkpoint can be stored on the portable storage device. At this point, the previous operation log can be deleted. As operations accumulate, so does the time to bring a new second-class replica up to date. This means that there exists a break-even point where it takes less time to simply create a new checkpoint from the first-class replica on the home server and download it over the Internet than it takes to instantiate a replica from portable storage.

## 4 Evaluation

In this section, we quantify the benefits of portable storage. We compare the time to instantiate a second-class replica on a new surrogate with and without portable storage. We also examine the storage requirements of some typical services.

### 4.1 Methodology

The client platform in our evaluation is an iPAQ 3970 handheld running the Linux 2.4.19-rmk3 kernel. The handheld has an XScale-PXA250 processor, 64 MB of DRAM. It uses a 11 Mb/s Cisco




This figure shows the topology for the experiment. The handheld user moves from the hotspot in the middle to the one on the left. We refer to the surrogate at the middle hotspot as the distant surrogate and the other as the nearby surrogate.

**Figure 3**. The logical topology for the experiment

350 802.11b card for network communication and a 4 GB Hitachi microdrive as portable storage. The home server is a Dell Precision 350 with a 3.06 GHz Pentium 4 processor running Red Hat 8 Linux. The nearby surrogate is a Dell Optiplex 370 with a 2.8 GHz Pentium 4 processor running Red Hat 9 Linux. The distant surrogate is an IBM X31 Thinkpad with a 1.6 GHz Pentium M processor.

The network topology between the home server, the distant surrogate, and the nearby surrogate is shown in Figure 3. We emulate this topology by routing packets through a computer running the NISTnet network emulator [3]. We assume surrogates are located in wireless hotspots with upstream and downstream bandwidth of 1.5 Mb/s. The home server is connected through a DSL link with upstream bandwidth of 256 Kb/s.

We show results for two services: speech recognition using IBM ViaVoice and the VNC remote desktop. The first is stateless and the second is stateful. We ran a repeating, fixed workload for each application. For speech, we recognized a pre-recorded phrase and then paused 5 seconds before the next iteration. For VNC, a program emulated a user opening a Word document, inserting text at the beginning, saving, and closing the document. The think time between each iteration was 10 seconds.

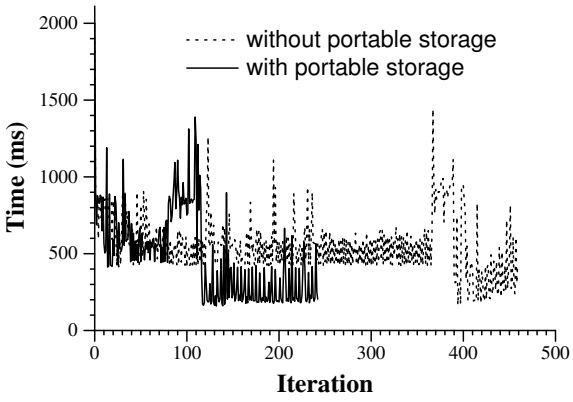
Both applications ran inside a VMware virtual machine configured with 4 GB of hard disk and 128 MB of memory. We installed both applications from a vanilla Windows XP image.

In our experiments, we examined how portable storage improves the instantiation of new replicas. The Hitachi 4 GB hard drive is inserted into the iPAQ throughout these experiments. As each experiment begins, the handheld user moves from the middle hotspot to the one located on the left in Figure 3. Thus, the user has a first-class replica running on the home server and a second-class replica running on the distant surrogate at the start of the experiment. After some time, Slingshot detects the existence of the nearby surrogate at the new hotspot and instantiates a replica on it.

### 4.2 Stateless service: speech recognition

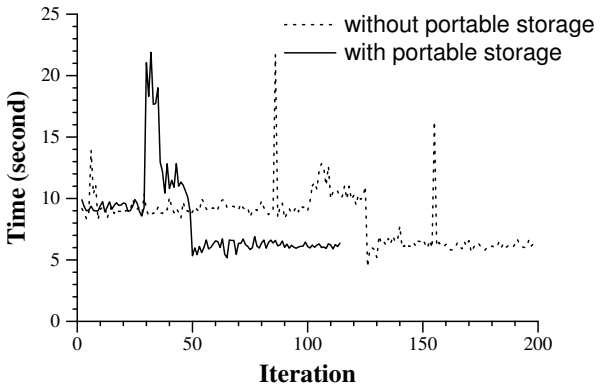
Figure 4 shows application response time during the instantiation of a new replica of the stateless speech service with and without portable storage. Note that the instantiation of a replica at the nearby wireless hotspot reduces application response time by 48% in both cases. In each case, Slingshot starts instantiating the replica after iteration 80. The time to instantiate a second-class replica without portable storage is 26:30 minutes. Portable storage reduces this time to 3:07 minutes, an 85% improvement. This means that the user enjoys improved response time for her application much sooner.

In Figure 4, between iterations 81 and 130 on the




This graph shows how response time varies during the instantiate of a speech recognition service

**Figure 4**. Speech recognition service




This graph shows how response time varies during the instantiate of a VNC remote desktop service

**Figure 5**. VNC remote desktop

solid portable storage line, there is a spike in response time. During this period, the new surrogate is fetching the volatile state from portable storage. This creates substantial network traffic that delays the transmission of remote requests to existing replicas, as well as the receipt of their replies. In the future, we plan to eliminate this behavior by prioritizing foreground application activity; however, this may slightly delay the instantiation of new replicas.

Figure 4 also shows a subtle benefit of portable storage. Without portable storage, application response time shows a spike before iteration 400. During this period, the newly instantiated replica on the nearby surrogate offers poor response time because it must fetch a considerable amount of its persistent state from the distant home server. Only after this state is fetched can it offer improved response time. However, if the new replica is able to fetch its persistent state from portable storage, then this data can be retrieved quicker, leading to better response time.

### 4.3 Stateful application: remote desktop

We next ran the same experiment for the stateful VNC remote desktop service. The results are shown in Figure 5, which compares application response time with and without portable storage. In both cases, we begin instantiating a new replica after 30 iterations. For this application, instantiating a replica at the nearby hotspot reduces response time by 35%. Without portable storage, the new replica is ready after 30:40 minutes at iteration 125. This includes 23:08 minutes to transfer state and 7:24 minutes to replay the log. Portable storage decreases the instantiation time to 7:20 minutes, which is comprised of 2:50 minutes of transfer time and 4:30 minutes of replay time. Overall, portable storage reduces the time to instantiate the new replica by 75%, despite having to replay the first 30 iterations of logged activity. In other words, the time to replay logged events is much less than the extra time that is needed to fetch an up-to-date checkpoint from the home server. We estimated the break-even point for this application to be roughly 210 iterations, or approximately 70 minutes of application activity. This indicates that the log size can grow to be quite large before it makes sense to fetch a new checkpoint over a wide-area connection.

### 4.4 Storage requirements

For the speech recognition service, the compressed volatile state is 36 MB and the persistent state is 2 GB. For the VNC service, the compressed volatile state is 30 MB and the persistent state is 1.5 GB. However, during the execution of our experiments the speech replica only fetched 6 MB of data chunks from the persistent storage, while the VNC replica fetched 8 MB. This represents less than 0.5% of the persistent state of each service. These results indicate that we could significantly reduce the storage requirements for Slingshot services if we could accurately predict which chunks will be needed.

We also examined the overlap in persistent state between the two services. In total, 1.17 GB of data is common to the two services—primarily because

they both run on a common Windows XP platform. This means that content-addressable storage can reduce the combined storage requirements of the two services by 33% from 3.5 GB to 2.33 GB.

## 5 Related Work

To the best of our knowledge, Slingshot is the first system to dynamically instantiate replicas of stateful applications in order to improve the performance of small, resource-poor mobile computers. Slingshot is an instance of cyber foraging [1], the opportunistic use of surrogates to augment the capabilities of mobile computers. Previous work in Spectra [4] examined how a cyber foraging system could locate the best server and application partitioning to use given dynamic resource constraints. In contrast, Slingshot takes this selection as a given and provides a mechanism for utilizing surrogate resources. More recently, Balan [2] and Goyal [6] have also proposed cyber foraging infrastructure. Compared to these systems, the major capability added by Slingshot is the ability to execute stateful services on surrogate computers.

Tolia et al. [11] use lookaside caching to integrate portable storage into a distributed file system and support Internet Suspend/Resume, which allows a user to migrate their computing environment between two desktop computers. In contrast, our work targets mobile computers such as handhelds and uses portable storage to support cyber foraging.

## 6 Conclusion and Future Work

The results in this paper show that portable storage can decrease the time to instantiate a new replica by up to 85% in a cyber foraging infrastructure. Further, we show how stateful applications can benefit from portable storage by logging and deterministically replaying remote operations. Finally, we quantify the storage requirements for two services.

Our future work will focus on prioritizing wireless network traffic so that the background traffic associated with replica instantiation does not adversely impact the foreground traffic associated with application activity. We also plan to investigate methods for predicting which chunks of a service's persistent state are most likely to be accessed by a surrogate. These predictions can be used to prioritize which chunks are cached on portable storage.

## References


[1] BALAN, R., FLINN, J., SATYANARAYANAN, M., SINNAMOHIDEEN, S., AND YANG, H.-I. The case for cyber foraging. In *the 10th ACM SIGOPS European Workshop* (Saint-Emilion, France, September 2002).

[2] BALAN, R. K., SATYANARAYANAN, M., PARK, S., AND OKOSHI, T. Tactics-based remote execution for mobile computing. In *Proceedings of the 1st Annual Conference on Mobile Computing Systems, Applications and Services* (San Francisco, CA, May 2003), pp. 273–286.

[3] CARSON, M. *Adaptation and Protocol Testing thorugh Network Emulation*. NIST, http://snad.ncsl.nist.gov/itg/nistnet/slides/index.htm.

[4] FLINN, J., NARAYANAN, D., AND SATYANARAYANAN, M. Self-tuned remote execution for pervasive computing. In *Proceedings of the 8th Workshop on Hot Topics in Operating Systems (HotOS-VIII)* (Schloss Elmau, Germany, May 2001), pp. 61–66.

[5] FLINN, J., SINNAMOHIDEEN, S., TOLIA, N., AND SATYANARAYANAN, M. Data staging for untrusted surrogates. In *Proceedings of the 2nd USENIX Conference on File and Storage Technology* (San Francisco, CA, March/April 2003), pp. 15–28.

[6] GOYAL, S., AND CARTER, J. A lightweight secure cyber foraging infrastructure for resource-constrained devices. In *Proceedings of the 6th IEEE Workshop on Mobile Computing Systems and Applications* (Lake Windermere, England, December 2004).

[7] KOZUCH, M., AND SATYANARAYANAN, M. Internet Suspend/Resume. In *Proceedings of the 4th IEEE Workshop on Mobile Computing Systems and Applications* (Callicoon, NY, June 2002).

[8] RODRIGUES, R., CASTRO, M., AND LISKOV, B. BASE: Using abstraction to improve fault tolerance. In *Proceedings of the 18th Symposium on Operating Systems Principles (SOSP)* (Banff, Canada, October 2001), pp. 15–28.

[9] SAPUNTZAKIS, C. P., CHANDRA, R., PFAFF, B., CHOW, J., LAM, M. S., AND ROSENBLUM, M. Optimizing the migration of virtual computers. In *Proceedings of the 5th Symposium on Operating System Design and Implementation* (Boston, MA, December 2002), pp. 377–390.

[10] SU, Y.-Y., AND FLINN, J. Slingshot: Deploying stateful services in wireless hotspots. submitted for publication.

[11] TOLIA, N., HARKES, J., KOZUCH, M., AND SATYANARAYANAN, M. Integrating portable and distributed storage. In *Proceedings of the 3rd Annual USENIX Conference on File and Storage Technologies* (San Francisco, CA, March/April 2004).

[12] TOLIA, N., KOZUCH, M., SATYANARAYANAN, M., KARP, B., BRESSOUD, T., AND PERRIG, A. Opportunistic use of content addressable storage for distributed file systems. In *Proceedings of the 2003 USENIX Annual Technical Conference* (May 2003), pp. 127–140.

[13] VMWARE, INC. *http://www.vmware.com*.

[14] WALDSPURGER, C. A. Memory resource management in VMware ESX server. In *Proceedings of the 5th Symposium on Operating Systems Design and Implementation* (Boston, MA, December 2002), pp. 181–194.