

The Case for Cyber Foraging

M. Satyanarayanan^{†‡}, Rajesh Balan[†], Shafeeq Sinnamohideen^{†‡}, Jason Flinn^{†‡} and Hen-I Yang[†]
[†]Carnegie Mellon University and [‡]Intel Research Pittsburgh

Abstract

Mobile devices are becoming increasingly smaller and lighter. However, this is done at the expense of their computational capability. Does this mean that mobile user will be unable to run computationally intensive applications? Cyber foraging construed as ‘living off the land’, may be an effective way to solve this dilemma. We propose the use of surrogates to ease the computational burden of mobile devices. In this paper, we introduce the research questions raised by cyber foraging and outline our work in developing staging servers and remote execution mechanisms using surrogates to improve the performance of mobile devices.

1 Introduction

The designers of mobile computing devices face a never-ending dilemma. On the one hand, size and weight are dominant factors. The need to make mobile devices smaller, lighter and run longer compromises their computing capabilities. On the other hand, user appetites are whetted by their desktop experiences. Meeting their ever-growing expectations requires computing and data storage ability beyond that of a tiny mobile computer with a small battery. How do we reconcile these contradictory demands?

We conjecture that *cyber foraging*, construed as “living off the land”, may be an effective way to solve this dilemma [4]. The idea is to dynamically augment the computing resources of a wireless mobile computer by exploiting compute servers and data staging servers that are located nearby. Such infrastructure may be discovered and used opportunistically at different locations in the course of a user’s movements. When no such infrastructure is available, the mobile computer offers a degraded but acceptable user experience. Using higher-level knowledge, it may also identify nearby locations that might offer a better user experience.

Who will provide the infrastructure for cyber foraging? Desktop computers at discount stores already sell today for a few hundred dollars, with prices continuing to drop. In the foreseeable future, we envision public spaces such as airport lounges and coffee shops being equipped with compute servers or data staging servers for the benefit of customers, much as comfortable chairs and table lamps are provided today. These will be connected to the wired Internet through high-bandwidth networks.

When hardware in the wired infrastructure plays this

role, we call it a *surrogate* of the mobile computer it is temporarily assisting. Two important attributes of surrogates are that they are *untrusted* and *unmanaged*. These are key assumptions because they reduce the total cost of ownership of surrogates and hence encourage their widespread deployment. It is the responsibility of mobile clients to establish adequate trust in the surrogates they choose to use.

In this paper we explore the concept of cyber foraging and discuss the research challenges posed by it. We also describe the status of our research in this area. Specifically, we illustrate how the use of surrogates can help in two distinct situations. First, we demonstrate how it can reduce cache miss service times in mobile file access. Second, we show how it enables compute-intensive applications like language translation and augmented reality to run on mobile hardware.

2 Usage Scenario and Research Challenges

We envision a typical scenario as follows. When a mobile computer enters a neighborhood, it first detects the presence of potential surrogates and negotiates their use. Communication with a surrogate is via short-range wireless technology. When an intensive computation accessing a large volume of data has to be performed, the mobile computer ships the computation to the surrogate; the latter may cache data from the Internet on its local disk in performing the computation. Alternatively, the surrogate may have staged data ahead of time in anticipation of the user’s arrival in the neighborhood. In that case, the surrogate may perform computations on behalf of the mobile computer or merely service its cache misses with low latency by avoiding Internet delays. When the mobile computer leaves the neighborhood, its surrogate bindings are broken, and any data staged on its behalf are discarded.

This usage scenario exposes many important research questions. Here are some examples:

- What is the system support needed to make surrogate use seamless and minimally intrusive for a user? What parts of this support are best provided by the mobile client, and what by the infrastructure?
- How much advance notice does a surrogate typically need to act as an effective staging server? Is this on the order of seconds, minutes or tens of minutes? What implications does this requirement have for the other components of a pervasive computing system?

- How does one establish an appropriate level of trust in a surrogate? What are useful levels of trust in practice? How applicable and useful is the concept of caching trust [3]? Can one amortize the cost of establishing trust across many surrogates in a neighborhood?
- How is load balancing on surrogates done? Is surrogate allocation based on an admission control approach, or a best-effort approach? How relevant is previous work on load balancing on networks of workstations?
- What are the implications for scalability? How dense does the fixed infrastructure have to be to avoid overloads during periods of peak demand?
- How does one discover the presence of surrogates? Of the many proposed service discovery mechanisms such as JINI, UPnP, and Bluetooth proximity detection, which is best suited for this purpose? Can one build a discovery mechanism that subsumes all of them for greatest flexibility?
- How do we deal with unmanaged surrogates? If they are used for remote execution, how can a mobile client ensure that the remote executing environment is correctly configured? What role, if any, can virtual machines such as VMware [2] play in establishing suitable execution environments? Can virtual machines also help with establishing trust?

This is obviously a very broad and ambitious research agenda, and our current work only addresses a subset of these questions. We summarize the status of our research in the rest of the paper. Section 3 describes how we are using data staging on surrogates to reduce the latency of servicing cache misses. Section 4 discusses our use of surrogates for remote execution. Note that trust is an issue that we have addressed in the context of data staging, but not in the context of remote execution. Section 5 describes our platform-independent approach to surrogate discovery.

3 Using Surrogates for Data Staging

3.1 Background

The goal of data staging is to reduce the latency of servicing cache misses in distributed file systems. A lower bound on this latency is the end-to-end network latency. A coast-to-coast ping in the United States typically takes about 60 ms., well above the speed-of-light bound of about 32 ms. The growing use of firewalls adds per-packet processing, and hence delay, to each packet. The impact of latency on distributed file systems is easily seen in interactive file-intensive applications such as mail readers, directory browsers, and digital photo albums. In such applications, a flurry of serial cache misses on relatively small files can result in annoying delays and sluggish behavior. Merely improving bandwidth does not help such interactive applications because they are latency-limited rather than bandwidth-limited.

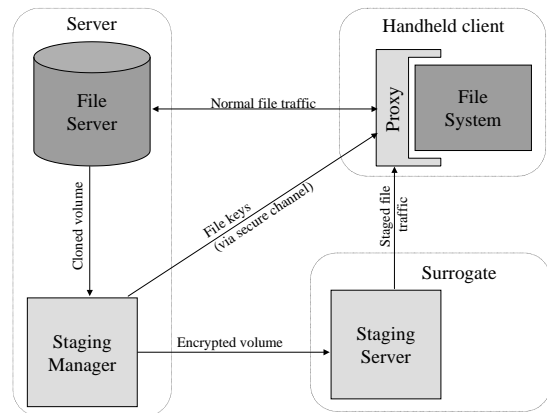


Figure 1. Data staging architecture

Although systems such as Coda [5] have shown that hoarding can reduce cache misses and hence mask network latency, there are limits to its effectiveness. First, size and weight restrictions may result in the flash memory or disk on a mobile computer being too small to hoard all relevant data. Second, some files that were not hoarded may unexpectedly become relevant to the user. Third, some files may be updated by other users and thus result in cache misses when accessed after a period of disconnection. These circumstances all lead to unavoidable cache misses and hence poor interactive performance.

Data staging helps solve this problem by speculatively prefetching distant data to nearby surrogates. In effect, clients borrow the storage capacity of surrogates and use it as a secondary file system cache. Cache misses from the mobile client are serviced by a *staging server* running on the nearby surrogate rather than by the distant file server.

3.2 Current Status

We have built an initial implementation of data staging for Coda. Figure 1 shows how our architecture is split across three computers: the file server (unmodified Coda server), the surrogate, and the mobile client. The client and surrogate are located close together and are typically connected by a low-latency wireless connection such as 802.11, Bluetooth or infrared. The file server is distant, so network communication to and from the file server incurs high latency. A proxy located on the client intercepts and redirects file system traffic. If a request is for data contained on a nearby staging server, the proxy directs the request to it. Otherwise, it forwards the request to the distant file server.

Since surrogates are untrusted, they never store staged data in the clear. A server encrypts a file before transmitting it to a surrogate, and it is decrypted only on the client as part of servicing a cache miss. We currently use DES private key encryption. Clients hoard encryption keys for all data of

potential interest. They also hoard MD5 checksums of that data to verify the integrity of files provided by the surrogate. Since storage requirements are small (at most 72 bytes per file), it is feasible to hoard keys and checksums even when the data itself is much too large to hoard.

We have striven to make surrogates as simple to manage as possible. The staging server is implemented using commodity software: we currently use an unmodified Apache Web server. The staging server maintains no critical long-term state. In case of a crash, data can be restaged from a server with no ill effects other than temporary performance degradation of cache miss handling.

3.3 Work in Progress

Our current implementation stages data an entire volume at a time. Even when volumes contain hundreds of megabytes of files, our measurements show sizeable benefits for clients, particularly if advance notice is available. The data on the staging server is also not kept consistent with updates made on the real server. We are working on removing both of these restrictions. Effectively, the client cache manager will manage its allocated space on the staging server as an extension of its local cache. Topics we plan on exploring are :

- Should the extended cache be exclusive or inclusive of the local cache ?
- What is the right cache management scheme to use ?
- Is hoarding an adequate method of preloading the staging server ?

Short para here on where we are headed

4 Using Surrogates for Remote Execution

4.1 Background

Remote execution for pervasive computing must reconcile multiple, possibly contradictory, goals. For example, executing a code component on a remote server might reduce client energy usage at the cost of increasing execution time. Also, due to the dynamic nature of the environment, it is not feasible to use static policies to determine how and where to remotely execute applications as the current resource situation may obsolete any statically chosen policy. Hence, the decision of how and where to remotely execute a code component must be determined dynamically based on the current resource availability in the environment.

Our work addresses the following aspects of remote execution:

- Monitoring resource availability
- Making dynamic remote execution decisions based on resource availability

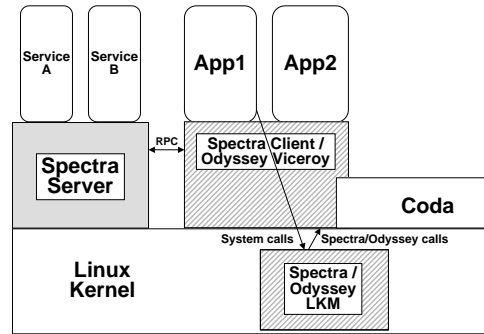


Figure 2. Spectra Architecture

- Simplifying the task of modifying applications to use remote execution

We currently trust surrogates used for remote execution. Developing the mechanisms for establishing this trust remains future work.

4.2 Current Status

We have implemented a system called Spectra [1] that monitors the current resource availability and dynamically determines the best remote execution plan for a given application. To make this decision, Spectra measures the supply and demand for many different resources such as bandwidth, CPU, and battery life. Figure 2 illustrates Spectra’s architecture.

Spectra targets applications that perform relatively coarse-grained operations of a second or more in duration. These applications include speech recognition, augmented reality, and language translation. Spectra tries to meet the application goals in the light of available resources. However, application goals can frequently be contradictory. For example, an application might require a high network throughput (which requires excessive use of a power hungry wireless network card) while also requiring low battery usage. Thus, Spectra has to reconcile these contradictory goals when making a remote execution decision.

To make good decisions, Spectra has to be able to predict the resource usage of alternative execution plans for an application. It does this through an approach called *self-tuning*, where it records the history of resource usage by an application and uses machine learning techniques to develop a prediction model.

4.3 Work in Progress

The major drawback of Spectra is that it requires application developers to explicitly modify their applications to use Spectra. This hurts software maintainence and portability. We are currently building a new remote execution system, called Chroma, that subsumes the functionality of

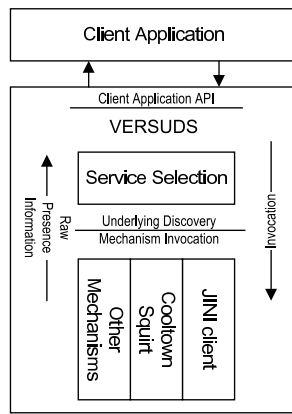


Figure 3. VERSUDS Architecture

Spectra. Chroma addresses the shortcomings of Spectra by separating the adaptive policies of an application from the actual decision making and enforcing of the policy at run-time.

Chroma is based on three observations derived from experience with Spectra:

- First, most applications for mobile devices can be created by modifying existing applications rather than writing new applications from scratch.
- Second, the modifications for adaptation typically affect only a small fraction of total application code size. Much of the complexity of implementing adaptation lies in understanding the base code well enough to be confident of the changes to make.
- Third, the changes for adaptation can be factored out cleanly and expressed in a platform-neutral manner.

Based on these observations, Chroma focuses on reducing the effort required to make applications use remote execution. It consists of three parts:

- A lightweight semi-automatic process for customizing the adaptation API used by the application. Such customization is targeted to the specific adaptation needs of each application.
- A tool for automatic generation of code stubs that map the customized API to the specific adaptation features of the underlying mobile computing platform.
- Run-time support for monitoring resource levels and triggering adaptation is factored out of applications into a set of operating system extensions for mobility.

5 Discovering Surrogates

The previous two sections have detailed our work in using surrogates for staging data and for remote execution. However, before either of these two operations can be performed, it is necessary to first discover the presence of surrogates in the current environment.

We thus implemented a versatile surrogate discovery service (VERSUDS) that uses existing service discovery mechanisms to detect the presence of surrogates in the proximity. VERSUDS provides a virtual layer that sits on top of these existing service discovery mechanisms. It provides a standardized API to applications and thus isolates applications from having to deal with different service discovery mechanisms as the environment changes. VERSUDS automatically translates application requests to the format of the underlying service discovery mechanism and vice versa. The VERSUDS architecture is shown in Figure 3.

VERSUDS currently provides support for JINI and Cooltown. It also supports application provided filters that specify which resources the application is interested in discovering. These filters can be used to specify dynamic system resource attributes such as CPU utilization and available memory as well as specific static attributes such as administrative domain and service price. We are currently extending its capability to support more complex application specific filters and to support other service discovery mechanisms.

6 Conclusion

We have described our initial research in the area of cyber foraging. Our work in developing data staging servers and remote execution systems has demonstrated the effectiveness of both in improving the performance of mobile devices. Our initial work in developing an environment independent surrogate discovery service has also been very encouraging and work in ongoing to improve the surrogate discovery service. In addition to the work in progress detailed in Sections 3.3 and 4.3, we are also planning on developing mechanisms to authenticate surrogates and establish trust relationships between surrogates and clients.

7 Acknowledgments

This research was supported by the National Science Foundation (NSF) under contracts CCR-9901696 and ANI-0081396, the Defense Advanced Projects Research Agency (DARPA) and the U.S. Navy (USN) under contract N660019928918. We would also like to thank Hewlett-Packard for donating notebooks to be used as surrogates and Compaq for donating handhelds to be used as clients. 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, DARPA, USN, HP, Compaq, nor the U.S. government.

References

- [1] FLINN, J. AND PARK, S. AND SATYANARAYANAN, M. Balancing Performance, Energy, and Quality in Pervasive Computing. In *Proceedings of the 22nd International Conference on Distributed Computing Systems* (Vienna, Austria, July 2002), vol. to appear.
- [2] GRINZO, L. Getting Virtual with VMware 2.0. *Linux Magazine* (June 2000).
- [3] SATYANARAYANAN, M. Caching Trust Rather Than Content. *Operating System Review* 34, 4 (October 2000).
- [4] SATYANARAYANAN, M. Pervasive Computing: Vision and Challenges. *IEEE Personal Communications* (July 2002).
- [5] SATYANARAYANAN, M. The Evolution of Coda. *ACM Transactions on Computer Systems to appear* (2002).