The Case for Cyber Foraging

Rajesh Balan[†,*], Jason Flinn[‡], M. Satyanarayanan[†‡], Shafeeq Sinnamohideen[†‡] and and Hen-I Yang[†]
[†]Carnegie Mellon University and [‡]Intel Research Pittsburgh
rajesh@cs.cmu.edu

Abstract

In this paper, we propose cyber foraging: a mechanism to augment the computational and storage capabilities of mobile devices. Cyber foraging uses opportunistically discovered servers in the environment to improve the performance of interactive applications and distributed file systems on mobile clients. We show how the performance of distributed file systems can be improved by staging data at these servers even though the servers are not trusted. We also show how the performance of interactive applications can be improved via remote execution. Finally, we present VERSUDS: a virtual interface to heteregeneous service discovery protocols that can be used to discover these servers.

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 long-running 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 [8]. The idea is to dynamically augment the computing resources of a wireless mobile computer by exploiting nearby compute and data staging servers. 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.

*School of Computer Science, Carnegie Mellon University, 5000 Forbes Avenue, Pittsburgh, PA 15213, USA

Who will provide the infrastructure for cyber foraging? Desktop computers at discount stores already sell 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 and 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. 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 show how data staging can reduce cache miss service times in mobile file access. Second, we show how remote execution 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 [7]? Can one amortize the cost of establishing trust across many surrogates in a neighborhood?
- How is load balancing on surrogates done? Is surrogate allocation to be done based on an admission control or best-effort approach? How relevant is previous work on load balancing on networks of workstations [2]?
- 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 [10], UPnP [6], and BlueTooth proximity detection [1], 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 [4] 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 yet in the context of remote execution. Section 5 describes our platform-independent approach to surrogate discovery.

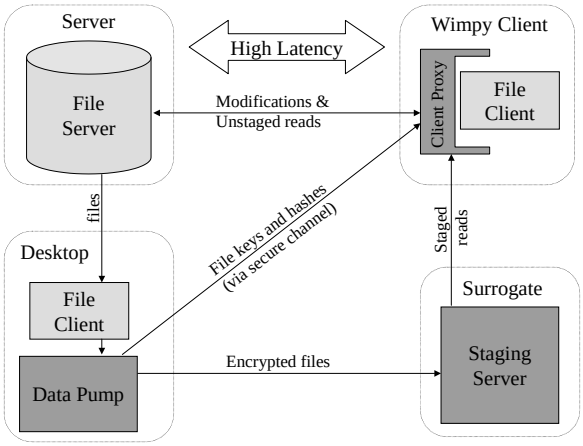

Figure 1. Data Staging Architecture

3. Using Surrogates for Data Staging

3.1. Background

Can an untrusted computer facilitate secure mobile data access? Surprisingly, the answer is "yes." Data staging enables untrusted, unmanaged computers to be used to improve the performance of cache miss handling in an Internet-wide distributed file system. The untrusted computer, called a *surrogate*, plays the role of a second-level file cache for a mobile client. By proactively staging data on the surrogate, cache misses from a nearby mobile client can be serviced at low latency (typically one wireless hop) instead of full Internet 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.

Although systems such as Coda [9] 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 four computers: the file server (unmodified Coda server), the surrogate, a home desktop machine, 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 the surrogate. Otherwise, it forwards the request to the distant file server.

Surrogates are untrusted—we therefore use end-to-end encryption to ensure privacy and secure hashes to guard against malicious modification of file data. The client proxy initiates the staging of data by contacting a data pump running on the end user's desktop machine. The client proxy specifies a list of files to stage—the data pump reads these files from the distributed file system, encrypts them with per-file keys, generates a secure hash of the data, and sends the encrypted files to the staging server. The data pump also sends the keys and hashes for the staged files to the client proxy using a secure channel. When an application reads data that is staged on the surrogate, the client proxy reads the file from the surrogate, decrypts it, verifies that it has not been modified using the secure hash, and returns the data to the application. 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.

Another important focus of our work is making surrogates as easy to manage as possible. We envision surrogates that are as simple as table lamps; they should require no system administrator or complex maintenance procedures.

We have identified two design principles that improve ease of management. The first principle is to build as much as possible upon widespread commodity software, so as to leverage the improved reliability that comes through the extensive testing provided by a large user community. To this end, we use the Apache Web server as the base system for our surrogates. We have identified the minimum set of additional functionality that must be located on the surrogate, and provide this functionality with CGI scripts. All other functionality is pushed to the client proxy and data pump in order to keep the custom code base on the surrogate as simple and reliable as possible.

The second design principle is to maintain no long-term state on the surrogate. For example, we do not buffer client modifications to file data on surrogates. Since all state is soft, no critical information is lost if the surrogate is disrupted by power failure or a system crash. The only consequence of surrogate failure is that clients receive the same sub-par performance for file accesses that they would have received if the surrogate had not been present in the first place.

3.3. Work in Progress

We have built a prototype of the data staging architecture that uses Coda as the base distributed file system. We are currently developing clients for both x86 (laptop) platforms, as well as StongArm (handheld) platforms. We have validated the prototype by replaying recorded traces of file system accesses—our results show that data staging can reduce the cumulative delay due to distributed file system accesses by up to 64%.

Our future work will explore the following questions:

- What is the best prediction algorithm for deciding which data to stage on a surrogate?
- What is the right cache management scheme?
- How will data staging benefit other distributed file systems such as NFS?

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.

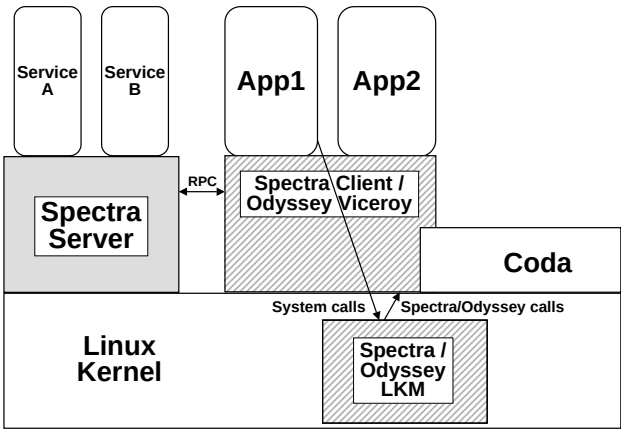

Figure 2. Spectra Architecture

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
- Simplifying the task of modifying applications to use remote execution
- Automatically using extra available resources in an over-provisioned environment to improve application performance

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 [3] 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, file cache state, 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 must 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 predict future usage.

4.2.1. Chroma

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

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

- Most applications for mobile devices can be created by modifying existing applications rather than writing new applications from scratch.
- 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.
- 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.

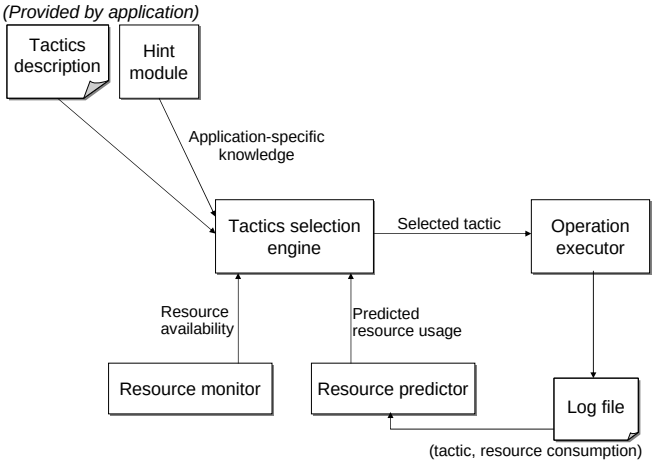

Figure 3. Chroma Run-time Components

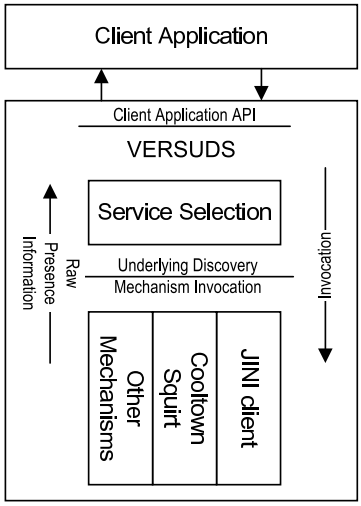

Figure 4. VERSUDS Architecture

4.2.2. Over-Provisioned Environments

Most remote execution systems are designed to operate in environments where resources are either scarce or just adequate. However, with the emergence of smart rooms and the dramatic decrease in the price of computing, environments where resources are over-provisioned are becoming a reality. These extra resources can be used to counter the variability inherent in mobile environments. When using just one remote server to perform an operation, any transient load spikes or network glitches affecting that server can have a dramatic effect on application performance. By using more than one server to perform the same operation, the effect of these load spikes and network glitches is minimized as the fastest result is returned to the application. Thus even if one server is experiencing transient load, other servers may be unloaded and will return a result faster than the loaded server (assuming all the servers are the same). One of the design goals of Chroma is to automatically use these extra resources to improve the performance of applications.

Chroma uses the notion of *tactics* to achieve this goal. Tactics are a concise declarative description of an applications remote execution capabilities and express the useful remote partitions of an application. Using this information, Chroma can automatically execute different parts of the application opportunistically on extra resources to achieve better application performance.

Figure 3 shows the run-time components used to handle tactics. Chroma determines the predicted resource usage of the tactics of the application by querying the resource prediction module. At the same time, Chroma determines the available resources via the resource measurement module. Chroma then decides how to remotely execute this application by finding the best match between available resources and the predicted resource usage of each tactic for an operation. Chroma also determines if extra resources are available in the environment. If they are, Chroma will opportunistically use these resources to remotely execute the application. We have shown that using tactics to opportunistically use extra resources in the environment can result in tremendous improvement in application response time.

4.3. Work in Progress

We are currently working on mechanisms to make Chroma even easier to use for application developers. Our use of a customized API coupled with automatic code generation is a major step in that direction. However, more work needs to be done in this area.

We also continuing our research in the use of tactics as a method of exploiting over-provisioned environments. We are investigating resource allocation polices that will ensure a fair use of these extra resources among multiple clients.

Finally, we are looking at existing work on security and integrity of remote servers. We hope to be able to use some of this work in our system.

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 have implemented a versatile surrogate discovery service (VERSUDS) that uses existing heterogeneous service discovery mechanisms to detect the presence of neighbour surrogates. 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 4.

VERSUDS currently provides support for JINI and Cooltown [5]. 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. We have shown that data staging is able to provide improved latency for file access without requiring the staging servers to be trusted. We have described our remote execution system, Chroma, and highlighted the key aspects that make Chroma easy to use for application developers. We have also described how Chroma is able to use extra resources in the environment to improvement application performance. Finally, we have described our initial work in developing an environment independent surrogate discovery service.

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. Rajesh Balan was additionally supported by a USENIX student research grant. We would also like to thank Hewlett-Packard for donating notebooks to be used as servers and Compaq for donating handhelds to be used as clients. Finally, we would like to thank Dushyanth Narayanan, SoYoung Park, Tadashi Okoshi, Bradley Schmerl and Joao Sousa for their many insightful comments and suggestions related to this work. 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, USENIX, Compaq, nor the U.S. government.

References

[1] 3COM, Agere, Ericsson, IBM, Intel, Microsoft, Motorola, Nokia and Toshiba. *Bluetooth Wireless Information Site*, 1999. http://www.bluetooth.com.

[2] A. C. Dusseau, R. H. Arpaci, and D. E. Culler. Effective Distributed Scheduling of Parallel Workloads. In *Proceedings of 1996 ACM Sigmetrics International Conference on Measurement and Modeling of Computer Systems*, Philadelphia, Pennsylvania, USA, May 1996.

[3] J. Flinn, S. Park, and M. Satyanarayanan. Balancing Performance, Energy, and Quality in Pervasive Computing. In *Proceedings of the 22nd International Conference on. Distributed Computing Systems*, Vienna, Austria, July 2002.

[4] L. Grinzo. Getting Virtual with VMware 2.0. *Linux Magazine*, June 2000.

[5] T. Kindberg, J. Barton, J. Morgan, G. Becker, I. Bedner, D. Caswell, P. Debaty, G. Gopal, M. Frid, V. Krishnan, H. Morris, C. Pering, J. Schettino, B. Serra, and M. Spasojevic. People, places, things: Web presence for the real world. In *Proceedings of the 3rd IEEE Workshop on Mobile Computing Systems and Applications (WMSCA 2000)*, Monterey, California, USA, Dec. 2000.

[6] Microsoft Corporation. *Universal Plug and Play Forum*, June 1999. http://www.upnp.org.

[7] M. Satyanarayanan. Caching Trust Rather Than Content. *Operating System Review*, 34(4), October 2000.

[8] Satyanarayanan, M. Pervasive computing: Vision and challenges. *IEEE Personal Communications*, 8(4), August 2001.

[9] Satyanarayanan, M. The Evolution of Coda. *ACM Transactions on Computer Systems*, 20(2), May 2002.

[10] J. Waldo. The Jini architecture for network-centric computing. *Communications of the ACM*, 42(7):76–82, 1999.