Distributed Virtual Machines: A System Architecture for Network Computing

Emin Gün Sirer, Robert Grimm, Brian N. Bershad, Arthur J. Gregory, Sean McDirmid
{egs,rgrimm,bershad,artjg,mcdirmid}@cs.washington.edu
http://kimera.cs.washington.edu
Dept. of Computer Science & Engineering
University of Washington
Seattle, WA 98195-2350

February 26, 1998

Abstract

Modern virtual machines, such as Java and Inferno, are emerging as network computing platforms. While today's virtual machines provide higher-level abstractions and more sophisticated services than their predecessors, and while they have migrated from dedicated mainframes to heterogeneous networked computers, their architecture has essentially remained intact. State of the art virtual machines are still monolithic, that is, all system components reside on the same host and are replicated among all clients in an organization. This crude replication of services among clients creates problems of security, manageability, performance and scalability.

We propose a distributed architecture for virtual machines based on *distributed service components*. In our proposed system, services that control security, resource management, and code optimization are factored out of clients and reside in enterprise-wide network servers. The services produce self-certifying, self-regulating, self-optimizing programs via binary rewriting. We are currently building a Java virtual machine based on this architecture. We argue that distributed virtual machine architectures enable higher integrity, manageability, performance and scalability than monolithic virtual machines where all components reside on all clients.

1. Introduction

Virtual machines have evolved significantly in the last two decades to emerge as the prevailing network computing platform. Modern virtual machines [Lindholm&Yellin96, Inferno, Adl-Tabatabai et al. 96] offer much more sophisticated services compared to their predecessors [IBMVM86]. Today's virtual machines (VMs) provide safety guarantees, dynamic extensibility, on-the-fly compilation, configurable security policies, and resource management facilities. Research trends indicate that these services will only grow in time [Wallach et al.97,Myers&Liskov97, Grimm&Bershad97]. Further, modern virtual machines are deployed in organizations with hundreds or thousands of hosts, in contrast with early systems that were typically confined to a few dedicated mainframes per enterprise. Nevertheless, the service architecture of virtual machines has remained static, even though virtual machine services have become much more numerous and complex, and even though the deployment style of VM systems has changed drastically. Today's virtual machines still rely on a monolithic architecture, in which all service components reside on the host computer, and are replicated across all virtual machines in an organization. Consequently, today's virtual machine systems suffer from security problems, are difficult to manage, impose high resource requirements and do not scale to large numbers of hosts.

The problems facing modern virtual machines stem from their monolithic architecture, which has resulted in virtual machines that are not modular, lack protection boundaries between components, and exhibit complex inter-component interactions. These attributes of monolithic systems combine to create problems of integrity, scalability, performance and manageability, which can be summarized as follows:

- Integrity: Since policy specification and security enforcement are performed on the same host which runs potentially untrusted applications, there is risk of long-term security compromises resulting from one-time security holes. Further, lack of address space boundaries between virtual machine components means that a flaw in a single component of the virtual machine places the entire machine at risk. Consequently, assuring the correctness of VM installations is a daunting task because the entire VM code base needs to be examined. The situation is analogous to the days before firewalls, when every networked host in an organization had to be protected against all bad packets that it might receive. The emergence of firewalls [Cheswick&Bellovin94] proved that it was simpler and more secure to concentrate functionality in a single packet-filter than to secure every host in an organization. The virtual machine situation today is identical, except that the services are considerably more complex than packet filtering.

- Manageability: Since each virtual machine is a completely independent entity, there is no central point of control in an enterprise. There are no transparent and comprehensive techniques for timely distribution of security upgrades, capturing audit trails, and pruning a network of rogue applications. To make matters worse, current system administration tools (e.g. rdist), push technologies (e.g. Marimba) and Internet protocols [RFC1157] do not support transparent management of virtual machines.
- Performance and Scalability: Virtual machine services, such as just-in-time compilation and verification, have substantial processing and memory requirements that can reduce overall application performance. Further, high resource requirements render virtual machines unsuitable for small, embedded hosts incapable of supporting all requisite components of a virtual machine.

2. Distributed Virtual Machine Architecture

We are building a virtual machine architecture that addresses the security, manageability, performance, and scalability requirements of network computing. These requirements are satisfied in our system by factoring virtual machine services into the smallest functionally independent units, and by migrating these services away from clients and onto enterprise-wide network servers. This approach reduces the size of the installed software base as well as the size of the trusted computing base for an organization. It further establishes physical boundaries between service components, isolates services from potentially untrusted applications, and reduces resource demands in clients.

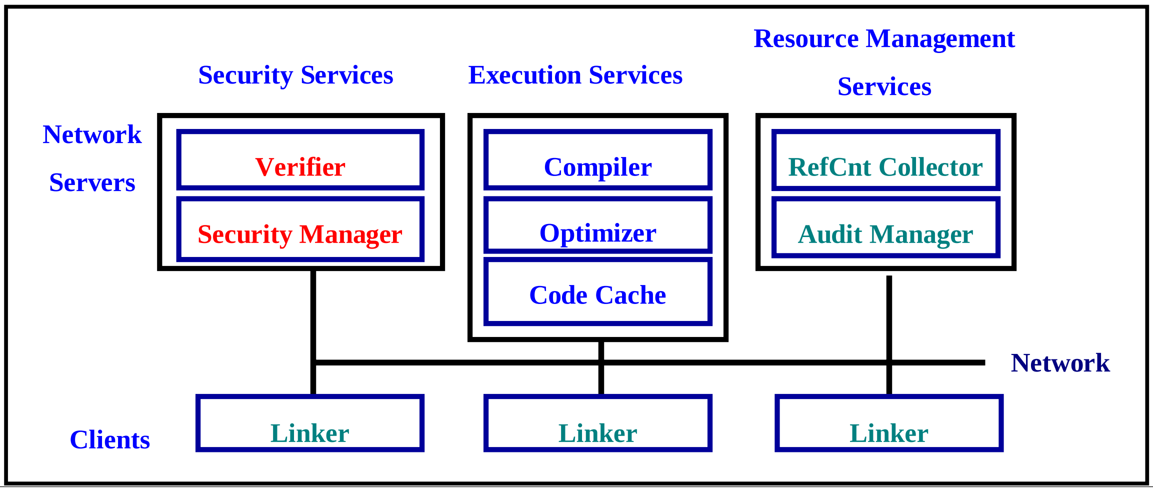

Figure 1. A distributed virtual machine, where services have been factored out of clients into network servers. Under a monolithic architecture, all clients would implement all of the VM services locally.

Figure 1 illustrates the architecture of a distributed virtual machine. Unlike a monolithic system, where each single client implements all of the virtual machine services, a distributed architecture concentrates services in network servers. The clients implement only dynamic linking to incorporate new code and cryptographic signature checking to ensure that applications have been vetted by the appropriate services. For example, an application that is introduced into a client without having passed through the security service would be barred from execution and redirected to the security server. A compilation server optionally translates applications into native code for virtual machines which do not support local compilation or interpretation. Overall, the enterprise-wide trusted computing base is smaller, and critical services are under direct administrative control.

2.1. Challenges

Distributing the functionality of a virtual machine across servers raises questions of how to structure and integrate the distributed components. There is a wide spectrum of possible approaches to service decomposition, ranging from monolithic services, through splitting services into policy and mechanism, to moving services completely out of clients. In this section, we discuss three interesting points in this spectrum, and make a case for service decomposition based on binary rewriting.

The simplest technique for distributing services is to locate them on remote hosts while retaining their existing interfaces via *remote service invocations*. For example, the security manager of a Java virtual machine, which is invoked at every system resource access, can be factored out by placing it on another host and by providing an RPC based invocation mechanism from the client runtime. This approach retains existing inter-component interfaces, and does not affect the service implementation. While this distribution technique has been used very successfully in systems built from scratch [Walker et al.83, Tanenbaum et al. 90], it has some drawbacks when applied to today's existing infrastructure and modes of interaction.

First, distribution based on remote service invocations requires full network connectivity, since the network services have to be contacted to perform critical system services such as resource usage authorization. Thus, this distribution model does not support a virtual machine that downloads an application from the network and later works in disconnected mode. Further, remote services incur communication overhead at run-time and will result in user-observed delays, especially when the clients are connected over low-bandwidth, high-latency links like modems. Finally, remote service implementations require that the VM be modified to convert local service invocations into remote procedure calls. Hence, they do not offer any benefits to the millions of already-deployed monolithic virtual machines in commercial browsers.

Fundamentally, monolithic systems have encouraged frequent, dynamic communication between services because communication has been easy and cheap. Consequently, distribution attempts that preserve the loose service boundaries and frequent runtime communication patterns found in monolithic virtual machines are faced with too much communication across too many components. This was the lesson learned in the 1980's as the systems community struggled to decompose monolithic operating systems into microkernel-based operating system servers [Accetta et al. 89]. Similarly, distributing functionality via remote service invocations, though simple because it retains existing service interfaces and implementations, poses problems of connectivity and latency.

One alternative distribution technique that addresses the shortcomings of remote service invocations is to move high-level service functionality into servers while leaving low-level, mechanistic service components on the clients. The advantage of such a *parameterized service* approach is that the communication between clients and servers is limited to the exchange of high-level service parameters. For example, in this model, a client would support a security enforcement mechanism that is driven by a compact policy specification supplied by a network server. While the low-level aspects of the system reside on local clients, and are therefore difficult to secure and manage, the high-level behavior can be controlled from a central network server. Such an approach is best suited to an environment where the low-level service components are small and simple, so that they do not consume excessive client resources and are not subject to frequent changes. In the case of Java, the low-level mechanisms are rather complex and do not easily yield to fine-grained decomposition. For instance, the verification service requires a complete dataflow analysis engine to perform its task, does not readily factor into components, and the history of the Java VM shows that verification is hard to get right [Kimera,Dean et al. 97]. Services such as security enforcement, compilation and optimization suffer similar problems with a parameterized approach. Finally, parameterized services require that virtual machines be outfitted with a new service implementation, and thus they do not benefit the existing base of monolithic virtual machines.

To address the deficiencies of remote service invocation and parameterized services, while still supporting them where they are applicable, we propose a model of service distribution based on *binary rewriting*. Under this model, individual services reside on network servers. They operate by manipulating the application code stream in such a way that the application becomes self-managing. For instance, a security enforcement service inspects the arguments of system calls before the execution of an application. Where the arguments cannot be determined statically, the service injects code into the application to perform the necessary check at run-time. Since the application is self-checking, a client can simply execute the code processed by the security manager without performing any checks of its own. This model is general enough to support embedding an entire service within the application, parameterizing the service into client-side mechanism and server-side policy, as well as utilizing traditional remote service invocations. It also provides a mechanism by which client-side code can be effectively managed throughout the organization. Structuring the virtual machine services around binary rewriting makes them applicable even to existing monolithic VMs. In the example above, a monolithic virtual machine may subject the rewritten application to redundant checks, but it also benefits from the centralized security service. Consequently, this arrangement provides a gradual conversion path from monolithic to distributed virtual machines.

2.2. An Example: Verification as a Standalone Service

As part of this project, we have built a standalone Java verifier that executes transparently on a separate proxy server to provide an additional level of protection for desktop browsers. The verifier ensures that incoming Java code conforms to a set of system safety axioms defined in the Java VM specification. Access logs for our web servers indicate that at least 1 in 5 deployed browsers contain well known verification flaws that can result in security breaches. We found that separating the verifier from the rest of the JVM facilitated a clean, robust implementation. The Java verifier is based on a generic Java rewriting engine, which we are using to build the other service components of a distributed virtual machine. We are currently in the process of building security, auditing, and compilation services. We have also built our own Java runtime to facilitate research with service placement.

3. Summary

We are building a computing infrastructure for enterprises that takes advantage of the portability and uniformity of virtual machines, while also providing secure, manageable, efficient and scalable services. A distributed virtual machine architecture achieves these goals since it reduces the installed and trusted computing base, moves critical functionality out of clients, and provides a point of network-wide control for the system administrator or the IT manager. Structuring the virtual machine services around binary rewriting makes them applicable even to existing monolithic clients, and provides a gradual conversion path from monolithic to distributed virtual machines.

4. References

[Accetta et al. 89] Accetta, M. J., Baron, R. V., Bolosky, W., Golub, D. B., Rashid R. F., Tevanian, A. and Young, M. W. “Mach: A New Kernel Foundation for Unix Development.” USENIX 1989.

[Adl-Tabatabai et al. 96] Adl-Tabatabai, A., Langdale, G., Lucco, S. and Wahbe, R. “Efficient and Language-Independent Mobile Programs." PLDI, May, 1996, p. 127-136.

[RFC1157] Case, J., Fedor, M., Schoffstall, M. and J. Davin, "Simple Network Management Protocol", RFC 1157, May 1990.

[Cheswick&Bellovin94] Cheswick, W. R. and Bellovin, S. Firewalls and Internet Security: Repelling the Wily Hacker. Addison-Wesley, 1994.

[Dean et al. 97] Dean, D., Felten, E. W., Wallach, D. S. and Belfanz, D. “Java Security: Web Browers and Beyond.” In Internet Beseiged: Countering Cyberspace Scofflaws, D. E. Denning and P. J. Denning, eds. ACM Press, October 1997.

[Grimm&Bershad97] Grimm, R. and Bershad, B. “Providing Policy-Neutral and Transparent Access Control in Extensible Systems.” Technical Report UW-CSE-98-02-02, February 1998.

[IBMVM86] IBM Corporation. Virtual Machine/System Product Application Development Guide, Release 5. Endicott, New York, 1986.

[Inferno] Lucent Technologies. Inferno. http://inferno.bell-labs.com/inferno/

[Kimera] Sirer, E. G., Gregory, A. J., McDirmid, S. and Bershad, B. The Kimera Project. http://kimera.cs.washington .edu/flaws/

[Lindholm&Yellin96] Lindholm, T. and Yellin, F. The Java Virtual Machine Specification. Addison-Wesley, 1996.

[Myers&Liskov97] Myers, A. C. and Liskov, B. “A Decentralized Model for Information Flow Control.” Proceedings of the Symposium on Operating System Principles, 1997.

[Tanenbaum et al. 90] Tanenbaum, A.S., Renesse, R. van, Staveren, H. van., Sharp, G.J., Mullender, S.J., Jansen, A.J., and Rossum, G. van: "Experiences with the Amoeba Distributed Operating System," Commun. ACM, vol. 33, pp. 46-63, Dec. 1990

[Wallach et al.97] D. S. Wallach, D. Balfanz, D. Dean and E. W. Felten. “Extensible Security Architectures for Java.” SOSP 1997.

[Walker et al.83] Walker, B., Popek, G., English, R., Kline, C. and Thiel, G. “The LOCUS Distributed Operating System.” Proceedings of the 1983 SOSP, 1983, p. 49-69.