AFRL-IF-RS-TR-2003-159
Final Technical Report
June 2003

OPERATING SYSTEM SERVICES FOR NETWORKED CLUSTERS

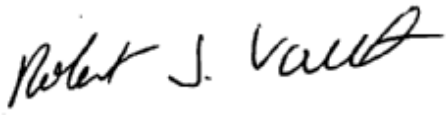
University of Washington

Sponsored by
Defense Advanced Research Projects Agency
DARPA Order No. F214

AIR FORCE RESEARCH LABORATORY
INFORMATION DIRECTORATE
ROME RESEARCH SITE
ROME, NEW YORK

This report has been reviewed by the Air Force Research Laboratory, Information Directorate, Public Affairs Office (IFOIPA) and is releasable to the National Technical Information Service (NTIS). At NTIS it will be releasable to the general public, including foreign nations.

AFRL-IF-RS-TR-2003-159 has been reviewed and is approved for publication.

APPROVED: 

ROBERT J. VAETH
Project Engineer

FOR THE DIRECTOR: 

WARREN H. DEBANY JR., Technical Advisor
Information Grid Division
Information Directorate

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 074-0188

1. AGENCY USE ONLY (Leave blank)	2. REPORT DATE	3. REPORT TYPE AND DATES COVERED
	JUNE 2003	Final May 97 – May 02

4. TITLE AND SUBTITLE
OPERATING SYSTEM SERVICES FOR NETWORKED CLUSTERS

5. FUNDING NUMBERS
C - F30602-97-2-0226
PE - 62301E
PR - F214
TA - 71
WU - 05

6. AUTHOR(S)
Brian N. Bershad and Henry M. Levy

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)
University of Washington
Grant and Contract Services
3935 University Way
NE Seattle Washington 98105-6613

8. PERFORMING ORGANIZATION REPORT NUMBER
N/A

9. SPONSORING / MONITORING AGENCY NAME(S) AND ADDRESS(ES)
Defense Advanced Research Projects Agency
3701 North Fairfax Drive
Arlington Virginia 22203-1714

AFRL/IFGB
525 Brooks Road
Rome New York 13441-4505

10. SPONSORING / MONITORING AGENCY REPORT NUMBER
AFRL-IF-RS-TR-2003-159

11. SUPPLEMENTARY NOTES
AFRL Project Engineer: Robert J. Vaeth/IFGB/(315) 330-2182/ Robert.Vaeth@rl.af.mil

12a. DISTRIBUTION / AVAILABILITY STATEMENT
APPROVED FOR PUBLIC RELEASE; DISTRIBUTION UNLIMITED.

12b. DISTRIBUTION CODE

13. ABSTRACT ***(Maximum 200 Words)***
The Network Clusters project explored several key areas in the design and deployment of large scale computing clusters, including: communication, security, and services. In the area of communication, we developed a new computing infrastructure for partitioning protocols between the primary processor and an embedded co processor. Our solution is unique in that it is designed to allow new protocols to be run on the coprocessor without compromising the safety of previously installed protocols. In the area of security, we developed a new architecture for specifying and enforcing security properties on code that runs on an individual machine. Moreover, we designed a new distributed systems platform for securing networks of virtual machines. In the area of services, we developed a highly scalable cluster-based Mail service, capable of processing over 1 billion mail messages per day.

14. SUBJECT TERMS
Network, Communication, Security, Protocols, Clusters, Virtual Machines, Computing Infrastructure, Design and Development of Computing Clusters

15. NUMBER OF PAGES
13

16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT	18. SECURITY CLASSIFICATION OF THIS PAGE	19. SECURITY CLASSIFICATION OF ABSTRACT	20. LIMITATION OF ABSTRACT
UNCLASSIFIED	UNCLASSIFIED	UNCLASSIFIED	UL

Table of Contents

Communication

Spine – Executing Protocols on Network Coprocessors

The emergence of fast, cheap embedded processors presents the opportunity to execute code directly on the network interface. We are developing an extensible execution environment, called SPINE that enables applications to compute directly on the network interface. This structure allows network-oriented applications to communicate with other applications executing on the host CPU, peer devices, and remote nodes with low latency and high efficiency.

Many I/O intensive applications such as multimedia client, file servers, host based IP routers often move large amounts of data between devices, and therefore place high I/O demands on both the host operating system and the underlying I/O subsystem. Although technology trends point to continued increases in link bandwidth, processor speed, and disk capacity the lagging performance improvements and scalability of I/O busses is increasingly becoming apparent for I/O intensive applications. This performance gap exists because recent improvements in workstation performance have not been balanced by similar improvements in I/O performance. The exponential growth of processor speed relative to the rest of the I/O system, though, presents the opportunity for application-specific processing to occur directly on intelligent I/O devices. Several network interface cards, such as the Myricom's LANai, Alteon's ACEnic, and I_2O systems, provide the infrastructure to compute on the device itself. With the technology trend of cheap, fast embedded processors (e.g., StrongARM, PowerPC, MIPS) used by intelligent network interface cards, the challenge is not so much in the hardware design as in a redesign of the software architecture needed to match the capabilities of the raw hardware.

We are working to move application-specific functionality directly onto the network interface, and thereby reduce I/O related data and control transfers to the host system to improve overall system performance. The resulting ensemble of host CPUs and device processors forms a potentially large distributed system.

In the context of our work, we are exploring how to program such a system at two levels. At one level, we are investigating how to migrate application functionality onto the network interface. Our approach is empirical: we take a monolithic application and migrate its I/O specific functionality into a number of *device extensions*. An extension is code that is logically part of the application, but runs directly on the network interface. At the next level, we are defining the operating systems interfaces that enable applications to compute directly on an intelligent network interface. Our operating system services rely on two technologies. First, applications and extensions communicate via a message-passing model based on Active Messages. Second, the extensions run in a safe execution environment, called SPINE, that is derived from the SPIN operating system.

Applications that will benefit from this software architecture range from those that perform streaming I/O (e.g., multimedia clients/servers and file-servers), host based IP routers, cluster

based storage management (e.g., Petal), to support for packet filtering (e.g., Lazy Receive Processing).

SPINE offers developers a software architecture for the following three features that are key to efficiently implement I/O intensive applications:

Device-to-device transfers. By avoiding extra copies of data, we can significantly reduce bandwidth requirements in and out of host memory as well as halving bandwidth over a shared bus, such as PCI. Additionally, intelligent devices can avoid unnecessary control transfers to the host system as they can process the data before transferring it to a peer device. Techniques, such as SPLICE, have been introduced to emulate the device-to-device transfer.

Host/Device protocol partitioning. Low-level protocol support for application-specific multicast, packet filtering (e.g., DPF) and quality of service (e.g., Lazy Receive Processing) has shown to significantly improve system performance.

Device-level memory management. An important performance aspect of a network system is the ability to transfer directly between the network interface and the application buffers. This type of support has been investigated by various projects (e.g., UTLB, AMII, and UNET/MM).

Using SPINE, we have demonstrated that intelligent devices make it possible to implement application-specific functionality inside the network interface. Although hardware designs using a "front-end" I/O processors are not new, they traditionally have been relegated to special purpose machines (e.g., Auspex NFS server), mainframes (e.g., IBM 390 with channel controllers), or supercomputer designs (e.g., Cray Y-MP).

We believe that current trends will continue to favor the split style of design reflected in SPINE. Two technologies though could challenge the soundness of the SPINE approach. First, I/O functions could become integrated into the core of mainstream CPUs -- an unlikely event given pressures for cache capacity. Second, a very low latency standard interconnect could become available. However, given that I/O interconnects by their very nature must be both open and enduring, we believe these non-technical forces hinder growth of I/O performance more than anything will.

Our two example applications show that many extensions are viable *even with an incredibly slow I/O processor*. Based on our experience with the LANai, we believe that more aggressive processor and hardware structures would have a large positive impact on performance. For example, hardware FIFOs could eliminate much of the coordination overhead in our current system. A faster clock rate alone would significantly improve the Active Message event dispatch rate as well. We expect that a system using a current high-end I/O processor (clocked at roughly 250 MHz and with a cache size of 16KB) could improve performance by a factor of five over our current system.

As embedded processors continue to increase in power relative to I/O rates, the number of extensions that are possible will greatly increase. For example, having several megabytes of memory on the device enables NFS and HTTP caching extensions. A vector unit would allow

many multimedia extensions. A faster CPU would allow the use of a virtual machine interpreter, enabling transparent execution of extensions regardless of the instruction set.

Next Generation Network Protocols -- Ipv6/Ipv4

IPv6 is a new version of the internetworking protocol designed to address the scalability and service shortcomings of the current standard, IPv4. Unfortunately, IPv4 and IPv6 are not directly compatible, so programs and systems designed to one standard can not communicate with those designed to the other. IPv4 systems, however, are ubiquitous and are not about to go away "over night" as the IPv6 systems are rolled in. Consequently, it is necessary to develop smooth transition mechanisms that enable applications to continue working while the network is being upgraded. In this paper we present the design and implementation of a transparent transition service that translates packet headers as they cross between IPv4 and IPv6 networks. While several such transition mechanisms have been proposed, ours is the first actual implementation. As a result, we are able to demonstrate and measure a working system, and report on the complexities involved in building and deploying such a system.

The current internetworking protocol, IPv4, eventually will be unable to adequately support additional nodes or the requirements of new applications. IPv6 is a new network protocol that features improved scalability and routing, security, ease-of-configuration, and higher performance compared to IPv4. Unfortunately, IPv6 is *incompatible* with IPv4 and to use the new protocol will require changes to the software in every networked device. IPv4 systems, however, are ubiquitous and are not about to go away "over night" as the IPv6 systems are rolled in. Consequently, it is necessary to develop transition mechanisms that enable applications to continue working while the hosts and networks are being upgraded. One suggested strategy is to translate IP headers as they cross between IPv4 and IPv6 networks. The requirement of header translation is to remain transparent to applications and the network.

In this paper we present two variations of IPv6/IPv4 translators that address these difficulties. The first variation uses *special* IPv6 addresses to easily translate packets transparently for all applications. Unfortunately, these special IPv6 addresses also require IPv6 routers to contain special routes to them, which is considered to be a bad idea because it creates more state for the router to maintain. The second variation maintains an explicit mapping between IPv4 and IPv6 addresses, and is therefore able to use standard IPv6 addresses that do not require any special treatment by IPv6 routers. Its drawback is that IP-addresses embedded in some applications' data stream, such as FTP, must be updated as well for the translation to be completely transparent.

We have built an IPv6/IPv4 network address and protocol translator as a device driver running in the Windows NT operating system. Our test environment consists of the translator as a gateway between IPv6 and IPv4 hosts connected to separate Ethernet segments, and it incurs little performance overhead. Between a pair of IPv6 and IPv4 nodes communicating via the translator, we have measured TCP bandwidth of 7210 Kbytes/second and roundtrip packet latencies of 424 microseconds over 100Mbit/second Ethernet links.

We have described the design and implementation of an IPv6/IPv4 network address and protocol translator, and briefly compared pros and cons of stateless vs. stateful translation. Our work subsumes both the stateless SIIT design and the stateful design. Despite the limitations of translation (e.g., loss of information) we believe that a translator can adequately fulfill the role of a short-term transition aid from IPv4 to IPv6, since it supports the majority of Internet traffic (HTTP, FTP, sendmail).

Based on our experience we conclude that an IPv6/IPv4 network address and protocol translator is complementary to the AIIH approach in transitioning from IPv4 to IPv6. In particular, we believe that it will be a valuable tool to developers porting applications from IPv4 to IPv6. For instance, a server application ported to IPv6 can be tested without having to port the client as well.

For more information about the IPv6/IPv4 translator, performance, and source availability please visit our web page at: www.cs.washington.edu/research/networking/napt

Security Architectures

Security Services for Extensible Platforms

Extensible systems, such as Java or the SPIN extensible operating system, allow for units of code, or extensions, to be added to a running system in almost arbitrary fashion. Extensions closely interact through low-latency but type-safe interfaces to form a tightly integrated system. As extensions can come from arbitrary sources, not all of whom can be trusted to conform to an organization's security policy, such structuring raises the question of how security constraints are enforced in an extensible system. In this paper, we present an access control mechanism for extensible systems to address this problem. Our access control mechanism decomposes access control into a policy-neutral enforcement manager and a security policy manager, and it is transparent to extensions in the absence of security violations. It structures the system into protection domains, enforces protection domains through access control checks, and performs auditing of system operations. The access control mechanism works by inspecting extensions for their types and operations to determine which abstractions require protection and by redirecting procedure or method invocations to inject access control operations into the system. We describe the design of this access control mechanism, present an implementation within the SPIN extensible operating system, and provide a qualitative as well as quantitative evaluation of the mechanism.

Extensible systems promise more power and flexibility than traditional systems and enable new applications such as smart clients or active networks. They are best characterized by their

support for dynamically com-posing units of code, called *extensions* in this paper. In these systems, extensions can be added to a running system in almost arbitrary fashion, and they interact through low-latency but type-safe interfaces with each other. We use the term "interface" in this paper to simply denote the types and operations exported by an extension; interfaces may declare types but are not, as in Java, type declarations themselves. Extensions and the core system services are typically collocated within the same address space and form a tightly integrated system. Consequently, extensible systems differ fundamentally from conventional systems, such as Unix, which rely on processes executing under the control of a privileged kernel.

As a result of this structuring, system security becomes an important challenge, and access control becomes a fundamental requirement for the success of extensible systems. Since system security is customarily ex-pressed through protection domains, an access control mechanism should —structure the system into protection domains (which are an orthogonal concept to conventional address spaces, -- enforce these domains through access control checks, and -- support auditing of system operations. Furthermore, an access control mechanism must address the fact that extensions often originate from other networked computers and are not trusted, yet execute as an integral part of an extensible system and interact closely with other extensions. In this paper, we present an access control mechanism for extensible systems that meets the above requirements. We build on the idea of separating policy and enforcement first explored by the distributed trusted operating system (DTOS) effort and introduce a mechanism that not only separates policy from enforcement, but also separates access control from the actual functionality of the system. The access control mechanism is based on a simple yet powerful model for the interaction between a policy-neutral enforcement manager and a given security policy, and it is transparent to extensions and the core system services in the absence of security violations.

Our access control mechanism works by inspecting extensions for their types and operations to determine which abstractions require protection and by redirecting procedure or method invocations to inject access control operations into the system. The access control mechanism provides three types of access control operations, which are expressed in terms of security identifiers, representing privilege, and permissions, representing the right to perform an operation. The operations are (1) explicit protection domain transfers to allow for a controlled change of privilege, (2) access checks to limit which procedures or methods can be invoked and which objects can be passed, and (3) auditing to provide a trace of system operations. The access control mechanism works at the granularity of individual procedures or methods and provides precise control over extensions and the core system services alike.

Our access control mechanism is based on the following three assumptions. First, because it is a software-only mechanism, it assumes that the code in an extensible system is safe, that is, that all code respects the declared interfaces and preserves referential integrity. Second, because our access control mechanism imposes access control operations on procedures and methods, it assumes that resources that need to be protected rely on encapsulation to hide their internal state. Finally, because our access control mechanism injects access control operations into an extensible system, it assumes the existence of some mechanism for binary interposition, such as the ability to dynamically patch object jump tables. The main contributions of this paper are twofold. First, based on the observation that extensible systems differ fundamentally from

conventional systems, we identify the specific goals for providing effective access control in extensible systems. Second, we present an access control mechanism that meets these goals by separating access control policy, enforcement, and functionality and which relies on a simple yet powerful model for their interaction.

Access control and its enforcement are but one aspect of the overall security of an extensible system. Other important issues, such as the specification of security policies or the expression and transfer of credentials for extensions, are only touched upon or not discussed at all in this paper. Furthermore, we assume the existence of some means, such as digital signatures, for authenticating both extensions and users. These issues are orthogonal to access control, and we believe that our access control mechanism can serve as a solid foundation for future work on other aspects of security in extensible systems.

Extensible systems, such as Java or the SPIN extensible operating system, have a different overall structure from conventional systems, such as Unix, because of their ability to dynamically compose units of code, or extensions. Extensions and the core services in an extensible system are typically collocated within the same address space and form a tightly integrated system, easily leading to more complex interactions between the different components than those in conventional systems. Since extensions generally cannot be trusted to conform to an organization's security policy, access control becomes a fundamental requirement for the success of extensible systems.

To protect extensions and the core services alike, access control for extensible systems needs to impose additional structure onto an extensible system. At the same time, it should only impose as much structure as is strictly necessary to preserve the advantages of extensible systems. Based on this realization, we have identified four goals to guide the design of an access control mechanism for extensible systems: (1) separate access control and functionality, (2) separate policy and enforcement, (3) use a simple yet expressive model, and (4) enforce transparently.

We have presented an access control mechanism for extensible systems that directly addresses these goals. Our access control mechanism separates access control and functionality by inspecting extensions for their types and operations to determine which abstractions require protection and by redirecting individual procedure or method invocations to inject access control operations into the system. It separates policy and enforcement by breaking up access control into a security policy manager, which makes the actual access decisions, and a policy-neutral enforcement manager, which enforces these decisions in the extensible system. It uses a simple yet expressive model that supports protection domain transfers to allow for a controlled change of privilege, access checks to limit which procedures or methods can be invoked and which objects can be passed, and auditing to provide a trace of system operations. Finally, it enforces transparently, as long as no violations of the security policy occur; extensions are notified of security faults so that they can implement their own failure model.

The implementation of our access control mechanism within the SPIN extensible operating system is simple and, even though the latency of individual access control operations can be noticeable, shows good end-to-end performance for a Web server benchmark. Based on our results, we predict that most systems will see a very small overhead for access control and thus

consider our access control mechanism an effective solution for access control in extensible systems.

Security for a Distributed Virtual Machine

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 component*s. 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.

Virtual machines have evolved significantly in the last two decades to emerge as the prevailing network computing platform. Modern virtual machines 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. 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 not trusted 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 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.

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. The 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.

Scalable Services for Networked Clusters

Porcupine

The Porcupine mail server, a cluster-based mail server, can handle up to 1 billion messages a day. Unlike common large-scale mail servers deployed today, there is no role separation among nodes. Each node in the cluster runs all the services supported by the cluster and balances the

workload dynamically using the cluster membership information. This architecture is more available, manageable, and scalable than traditional architecture.

Electronic mail service is one of the most valuable services offered by the Internet. E-mail traffic will continue to increase in the near future, due to the explosion in the number of users on the Internet and the increase in the number of messages each user receives. At present, sites such as AOL and Hotmail handle 10 million mail messages per day. Such giant email clearinghouses will become commonplace, and the amount of mail each of them handles will also grow in the future.

The goal of this project is to design and build an email server, called *Porcupine*, that can handle up to one billion messages a day from 10 million users, using a cluster of up to 1000 PCs. Porcupine takes advantage of today's cheap PCs and fast interconnects to achieve this goal without massive hardware or management expense. Porcupine treats the set of interconnected machines (the *cluster*) as one large server. Unlike traditional mail architectures, each node in the cluster performs a portion of all the functions that the cluster offers. For example, the mail user database is partitioned among nodes. User mailboxes are also distributed, and user messages can be stored on any node at any time. This design allows for high availability, high flexibility, and better load balancing.

Porcupine is designed to be cheap, scalable, fast and easy to manage. The simulation study shows that the system is indeed scalable, and failure recovery is not costly. The system is being implemented on a cluster of Linux PCs. In the next phase of the project, we plan to replicate both the user database and user mailboxes. Replication algorithms are still being designed. In addition, we plan to introduce a recipient-specific mail processing language used for junk mail rejection or mailing list processing. More information about Porcupine can be found at http://porcupine.cs.washington.edu.