社団法人 情報処理学会　研究報告
IPSJ SIG Technical Report

2004－OS－95　(7)
2004／2／27

SoftwarePot を用いた汎用仮想ホスティング

シュラーニ ペーテル[†1]　廣津　登志夫[†2,†3]　加　藤　和　彦[†4,†3]

近年のコンピュータの高速化により，一台のマシンを複数のホストのサーバにする仮想ホスティング技術が一般的に使われるようになったがこれにより以前よりもセキュリティを考慮することが必要となった．安全に仮想ホスティングを行うため広く利用されている仮想マシン技術は実行時オーバヘッドも大きく，セットアップにも手間がかかる．OS の機能として提供されている手法は実行は効率的だがセットアップの面では仮想マシンより改善がない．本論文では安全なソフトウェア実行システム SoftwarePot を用いた汎用仮想ホスティング手法を提案し，仮想ホスティング機能の実現について述べる．提案システムではディレクトリのマッピング機能により仮想ホスト環境のセットアップにかかる労力の軽減を実現している．

General Virtual Hosting with SoftwarePot

SURÁNYI, Péter ,[†1] Toshio Hirotsu [†2,†3] and Kazuhiko Kato[†4,†3]

While recent computers provide sufficient resources to share them between hosting several sites, network intrusions force people to focus on security. In order to realize safe virtual hosting, resources for hosted services need to be isolated. Virtual machine managers (VMMs) are commonly utilized for this purpose, however they require a significant amount of extra work for setting up and resources for running them. OS specific solutions such as FreeBSD jail provide more efficient execution, but still require the preparation of a full basic operating system for each virtual host. This paper proposes a general method for virtual hosting of arbitrary services based on the SoftwarePot Secure Execution System, and describes how the virtual hosting facility was implemented. Our approach allows reducing the work required for setting up the virtual hosting system.

1. Introduction

The amount and severity of network intrusions on the Internet is increasing day by day. Virtual hosting, ie. hosting services for several sites on the same machine not only increases the probability of that server being chosen as a target for an attack, but also widens the area of possible damage in the case that a security breach occured.

Security vulnerabilities are frequently discovered in server software, operating systems, and other pieces of software commonly employed on server machines. These weaknesses, if not dealt with promptly, may lead to a break-in possibly causing failure (downage) of the service or leakage, deletion or alteration of the data. Dynamic content created by the site author (e.g. CGI or other WWW scripts) that is not part of the server software itself may also contain security holes and make an otherwise secure server exposed to attacks.

In a virtual hosting scenario, with a single server hosting services for multiple sites, this means that having merely one service with a security vulnerability may gain access for the attacker to data belonging to all of the hosted sites. Even if there are no security holes in the server software at all, a single faulty user script may cause exposure of data of all of the hosted sites. Considering these factors, it can be easily seen that resource isolation is a vital problem in virtual hosting.

The aim of this research is to provide a means of virtual hosting that provides a high level of resource isolation together with ease of setup, while maintaining an acceptable level of runtime overhead. The ability to allow virtual hosting of existing services and design that allows portability are also considered as important aspects.

In this paper, we propose a system that provides resource isolation by running each service in a confined environment with a virtual file system view. This virtual view, besides preventing unwanted access to resources, can also help in absorbing differences between file system hierarchies of hosting machines.

The remainder of this paper is organized as follows.

†1 筑波大学大学院博士課程システム情報工学研究科
Graduate School of Systems and Information Engineering, University of Tsukuba
†2 日本電信電話株式会社 NTT 未来ねっと研究所
NTT Network Innovation Laboratories
†3 科学技術振興機構 CREST
CREST, Japan Science and Technology Agency
†4 筑波大学電子・情報工学系
Institute of Information Sciences and Electronics, University of Tsukuba

Section 2 discusses different methods employed for virtual hosting. Section 3 describes the SoftwarePot Secure Execution System. Section 4 explains how SoftwarePot was extended to make it suitable for virtual hosting. Section 5 evaluates the proposed system using application level benchmark. Section 6 concludes describes future work.

2. Related Work

At the time when the Internet was beginning to gain wider acceptance, virtually all sites were hosted on dedicated servers, with the single task of servicing requests for that site. However, with the computing power of servers increasing rapidly and the Internet expanding explosively, there was a strong demand to share resources between several sites.

Dedicated servers generally only had a single network interface with a single Internet address assigned to. Interface aliasing, i.e. assigning multiple Internet addresses to the same network interface, a technique now commonly utilized in virtual hosting, is also a relatively new concept. This means that many of the server programs still in use were not designed for being employed on a machine having multiple Internet addresses. These programs accept connections targeted to any of the addresses assigned to the machine, which can easily cause conflicts in an environment running several services for multiple hosts.

Virtual hosting was primarily realized by extending the server software to add support for running services for multiple Internet addresses. This was implemented in many World Wide Web (HTTP) and file (FTP) servers. While embedding virtual hosting support in the server software is arguably the most effective method for sharing resources, it also has its drawbacks. Adding support for multiple addresses increases server code complexity significantly as care must be taken to use the correct address both with incoming and outgoing connections. Since there is no way for the program to decide which of the multiple addresses available to use, it must be specified by the administrator. As different server programs usually require different ways of configuration, this can mean considerable extra work. This approach also brings up security concerns: a malicious attacker may cause failure of services, damage or leakage of information for any of the hosted sites by finding and exploiting a security vulnerability in merely a single one of the hosted sites (e.g. a faulty CGI script). To address these issues, recent trends favor running services for different virtual hosts in confined environments utilizing virtual machine monitors (VMMs) or isolation services provided by the operating systems.

Virtualization was not considered in the design for the predominant IA32 (also known as Intel x86 or i386) architecture. Because of this, solutions providing complete virtualization (e. g. VMware[6], VirtualPC) require a considerable amount of extra computing resources to achieve this goal, especially in the case of I/O intensive tasks. Besides the runtime overhead, another drawback of the virtual machine approach is that a separate copy of the operating system (guest OS) with its basic services needs to be installed and run within the virtual execution space. This requires additional work for setting up and increases memory and storage requirements as well.

Other VMMs such as Xen[2] and User Mode Linux(UML)[4] provide a limited level of virtualization that is sufficient for their goals. In the case of Xen, this avoids much of the runtime overhead, but it requires significant alteration of the guest OS code. UML's original goal is to run Linux within the user level of another Linux operating system. While originally created for aiding development of kernel code, UML is capable of virtual hosting, but having all I/O data pass through user-mode virtual device drivers increases runtime overhead considerably. These VMMs also don't solve the setup, memory and storage cost problems mentioned before.

UNIX and UNIX-like operating systems provide a system call named `chroot` that is capable to execute programs in a sandboxed file system environment, prohibiting access to all but a single subtree of the file system. Running network services in a `chroot` environment does avoid some security threats but it does not provide support for virtual hosting. To overcome this, version 4.0 of the FreeBSD operating system introduced a new system call named `jail` that provides a file system sandbox similar to that of `chroot`, with the additional feature of limiting the addresses used in network connections to a single one, thus making virtual hosting of practically any service possible. However, for `jail`, similarly to the virtual machine based approaches, it is necessary to install a copy of all the files that may be needed for the execution of the service. The FreeBSD manual suggests installing a comlete copy of the operating system inside the target directory. This requires a considerable amount of work to do for each service to host. Linux VServer[1] is a project aimed to provide services similar to that of `jail` by adding new system calls to the Linux operating system. This requires compiling a modified version of the kernel.

3. The SoftwarePot Secure Execution System

Installing and running programs on a system implies numerous security risks. Most recent applications consist of several files that need to be installed in the correct location in order for the program to function properly. This is usually performed by installer or package manager programs. In most cases installers are (and generally require to be) executed with super-user or administrator privileges, having full access to the machine's resources. This means that the installer is technically able to arbitrarily read, modify, overwrite or delete files, or initiate network transfers at will. Even for installers with no malicious intent it is often the case that some file to be installed is in conflict with a pre-installed file on the system, in which case either installing the new file or retaining the original one may cause one of the applications fail to operate correctly. In general, the user has to trust both the creator of the installer and the location it obtained it from, having no other way of ensuring that no unwanted effect is caused by the installation process.

SoftwarePot is designed for safely executing programs

that are not perfectly trusted by the user. This may include programs of untrusted source or programs that may be suspicable of security vulnerabilities.

3.1 Design

In SoftwarePot, similarly to most installers, the program and the related data files are stored together in a compressed data file, called *pot file*. However, contrary to the traditional installation methods, these files are not installed in the real target location, instead they are extracted in a temporary directory. During execution, the program is run in a special environment called *pot space*, that provides a virtual file system view, with the files in the archive mapped to their correct locations. The processes executing in this virtual space are called *pot processes*. Access to all other resources are governed by the system based on the security policy specification provided at execution time. This makes accessing external files possible by mapping them into the virtual file system view. Multiple pot files may share the same pot space, allowing non-essential data files or different versions of libraries to be distributed as separate pot files.

Operating the SoftwarePot system consists of using two commands. `makepot` creates a pot file based on a specification file describing files to incude in the archive, files and directories to map at runtime etc. `execpot` extracts and executes a pot file, optionally taking as argument an extra specification file that can specify new mappings and override the ones in the specification the pot file was created with.

The following is the sample specification file that can be used for generating a pot file with a single executable (`myserver`). In this example, the library `libc.so.6` is mapped into the pot space dynamically on the execution site.

```
<Skeleton>
  <Entry path="/mybin/myserver">
    <Arg>arg1</Arg>
  </Entry>
  <StaticFiles>
    <StaticFile vpath="/mybin/myserver">
      <Load protocol="local">
        <Path>/home/user/myserver</Path>
      </Load>
    </StaticFile>
  </StaticFiles>
  <DynamicFiles>
    <DynamicFile vpath="/lib/libc.so.6">
      <Load protocol="local">
        <Path>/lib/libc.so.6</Path>
      </Load>
    </DynamicFile>
  </DynamicFiles>
</Skeleton>
```

3.2 Implementation

In many operating systems, all access to system resources is performed via system calls. This means that all system resource references can be monitored by intercepting the corresponding system call(s). This is the approach taken in SoftwarePot to ensure safe execution of programs.

Figures 1 and 2 show the execution of a system call in the native system and in a SoftwarePot environment respectively. When running in SoftwarePot, system calls are intercepted, their arguments are checked and modified if necessary.

SoftwarePot is currently implemented in the following systems:

- Linux 2.4 / Intel IA32
- Linux 2.4 / ARM
- Solaris / SPARC

The Linux implementation currently requires loading a kernel module. This design has been chosen for performance reasons, it is believed to be possible to implement it utilizing the ptrace facility, allowing for an entirely user-space implementation. Recent improvements in the Linux ptrace facility may make such an implementation feasible.

3.3 Extensibility

SoftwarePot employs modular design in order to provide a high level of extensibility. Two kinds of modules can be used for extending SoftwarePot:

- Protocol modules
  SoftwarePot supports several methods for accessing files in the virtual view. It is possible for example for a file appearing in the virtual view of the Pot process to be retrieved through HTTP or other protocols dynamically when the process accesses the file. Protocol modules can be used to define these kinds of file access methods.
- Runtime modules
  Runtime modules can incorporate in SoftwarePot, intercept any system call and control the execution of the Pot process. This feature is employed in our implementation for virtual hosting.

4. Virtual Hosting with SoftwarePot

While SoftwarePot is an efficient and convenient system for running applications in confined environments, it was not designed for virtual hosting. Therefore, when applying SoftwarePot for virtual hosting, several problems arise.

4.1 Issues

In order to provide virtual hosting, we need to have multiple Internet Protocol (IP) addresses assigned to the machine. Several approaches are available for assigning addresses to virtual machines. VMware, User Mode Linux and Xen utilize a separate virtual network, bridging it to the actual network interface. This method provides a high level of separation at device level, however it requires forwarding packets between the virtual and the actual network interface. FreeBSD jail utilizes IP address aliases, reserving multiple IP addresses on the same interface. This method allows direct delivery of packets to the process by the kernel. This approach is taken in the proposed method as well, as it is believed to be the most efficient one.

Not separating virtual machines at a network device level means that all processes (including SoftwarePot

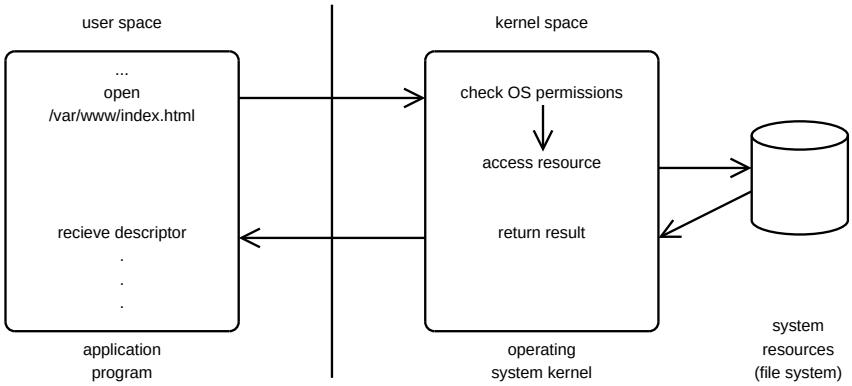

図 1 System call execution - normal flow

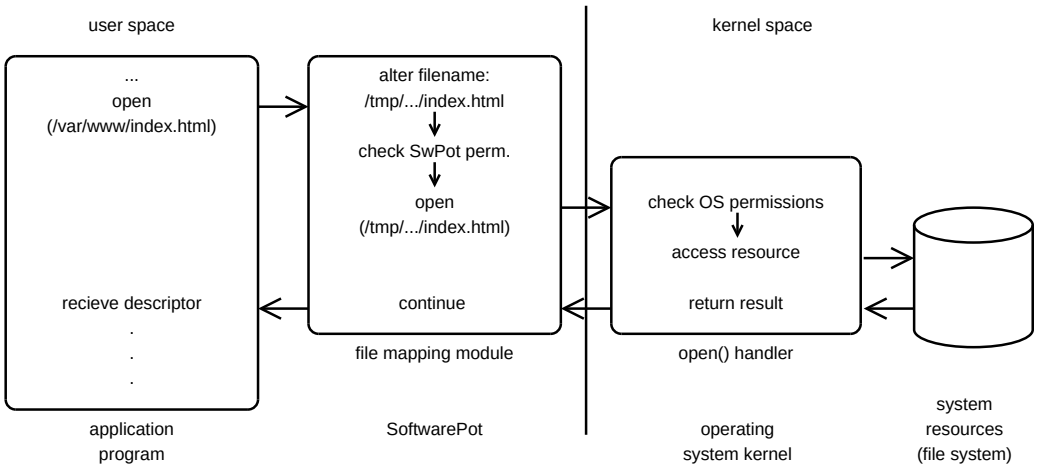

図 2 System call execution flow with SoftwarePot

processes) are able to use any of the IP addresses available. In order to use SoftwarePot processes for virtual hosting, this needs to be limited to the address of the virtual site.

Numerous server programs read the hostname of the machine in order to determine the IP address. In order to make these programs aware of their virtual IP addresses, it is neccessary to provide them with a virtual hostname.

4.2 Design

Server programs work by binding to a specific port number on the machine and waiting for clients to connect to that port. Binding to a port number is achieved by the `bind()` system call. While it is possible to specify an IP address to bind to, numerous server programs default to binding to the so-called "any" address (`INADDR_ANY`), which causes the program to recieve connections on all IP addresses assigned to the host. Several programs give an option for specifying the bind address explicitly. However, the means of configuration is different from server to server. In a virtual hosting scenario, it is often more desirable to handle the address configuration centrally in a unified way. ☆

Besides incoming connections, many server programs, most importantly proxy servers, also initiate connections to external hosts. When creating connections to other machines, the `connect()` system call is utilized. Unless the network socket has been bound to a specific address, which is generally not the case, the outgoing connection is assigned to a local address and port by the kernel. In virtual hosting, having the connection bound to an address that is different from the one assigned to the machine, may cause one of the following troubles.

- The connected host is misinformed about the origin of the connection, causing accesses to be associated with a different virtual host.
- If the machine is connected to several networks, depending on the routing configuration, this may allow clients from one network having the server make a

☆ xinetd[5] makes it possible to run several servers with a centralized configuration. This method, however, does not provide any isolation mechanisms between the services.

connection to the other network that is supposed to be separated from the client.

4.3 Implementation

To summarize the previous sections, the following main features need to be implemented in order to utilize SoftwarePot for virtual hosting.

- virtual IP address - forcing process to use assigned address
  - incoming connections (`bind()`)
  - outgoing connections (`connect()`)
- virtual hostname - report virtual hostname to the process

These functions are currently implemented in Linux on the Intel IA32 platform only, however they are believed to be easily ported to other UNIX-like OS's as well.

4.3.1 Incoming connections

The only thing required for IP address virtualization for incoming connections is to ensure that the argument passed to the `bind()` contains the IP address assigned to the virtual machine. This can be achieved by a simple SoftwarePot module that intercepts calls to `bind()` and rewrites the address in the argument to that of the virtual host. Additionally, for safety reasons, it is implemented so that it issues a warning when the address (before rewriting) neither matches the "Any" address nor the virtual host address. (The running program may be aware of its virtual address by performing a name server lookup on the virtual hostname.)

4.3.2 Outgoing connections

Implementing virtualization with regard to outgoing connections is a more sophisticated problem. Several approaches can be thought of in order to make sure that outgoing connections are bound to the correct address.

The address a connection is bound to is decided by the operating system kernel based on its routing tables and other internal data. The most straightforward method would be to check and ensure that these tables are correctly setup. However, this procedure would depend highly not only operating system, but, in the case of Linux, also its kernel version and configuration. Also, these kinds of low-level interactions with kernel structures are contrary to our goals of high portability and affinity with the original SoftwarePot architecture.

In order to achieve this goal in the system call level provided by the SoftwarePot infrastructure, it is necessary to make sure that the socket is bound to the local address before `connect()` is called. This could be achieved by forcing the pot-process to call `bind()` for each socket before a `connect()` call is made.

While the Reference Monitor Library makes it possible to intercept calls to `connect()`, by this point of time, the control of the process has been passed over to the kernel system call handler. This means that it is not possible to make the process invoke a `bind()` call before control has returned from the `connect()` call. This makes implementation as a single SoftwarePot module infeasible.

A different method is required for ensuring the binding of outgoing connections. As the interception point provided by SoftwarePot is in a position too late in the control flow, process needs to be overtaken in an earlier.

In UNIX systems, it is rarely the case that a system call is called directly from the code of an application program. Practically all programs access system calls via the wrapper functions of the same name provided in the C library. Also, it is important to note that almost all software for UNIX systems is made available in dynamically linked format[3]. Considering this, we decided to implement interception of outgoing connections using a technique called library interposition. By utilizing the library preload function of the loader,it is possible to preload a small wrapper library that defines a function called `connect()`. This causes the wrapper library function to be called every time the application makes a `connect()` call. The preloaded library can call `bind()` to bind the socket to the local address ☆, and then have the linker execute the original `connect()` code in the C library (which finally executes the system call with the same name). Figure 3 shows an example of how library interposition changes execution flow.

In order for the wrapper library to get preloaded, it is necessary to specify its path in either the `LD_PRELOAD` environmental variable or the `/etc/ld.so.preload` file. As environmental variables can be changed by user processes and are difficult to enforce, the proposed system is implemented by mapping a file containing its virtual path to `/etc/ld.so.preload`.

For almost all programs, this approach can intercept execution before entering the `connect()` system call. However in a security-conscious scenario as virtual hosting is, it is neccessary to consider all cases. The proposed system is not capable of detecting `connect()` calls invoked from statically-linked binaries, or any other code that bypasses the standard C library function call. To avoid security problems caused by these techniques, a violation check system has been implemented as a separate SoftwarePot module. It takes track of all `connect()`, `bind()` and other socket manipulation system calls and keeps a list of sockets bound to the local IP address. Every time `connect()` is called, it checks if the corresponding socket is on the list of bound ones and in the case it is not, denies execution of the system call.

It is important to note that many server programs don't connect to external hosts or only create connections for looking up hostnames. For these programs using this module and library is not necessary, network safety can be ensured by disabling `connect` completely or restricting to only allow creating connections targeted to the Domain Name Server.

4.3.3 Virtual Hostname

In order to obtain the hostname of the machine they're running on, most programs utilize `gethostname()`. In Linux, this is a C library function that is implemented so that it calls the `uname()` system call to fetch the hostname from the kernel. The virtual hostname feature is implemented by a module intercepting the `uname` system call.

☆ The library need not be aware of the local address. It is sufficient to give the "any" address as a parameter for bind(), as our SoftwarePot module will rewrite it to the virtual host addess before it is executed.

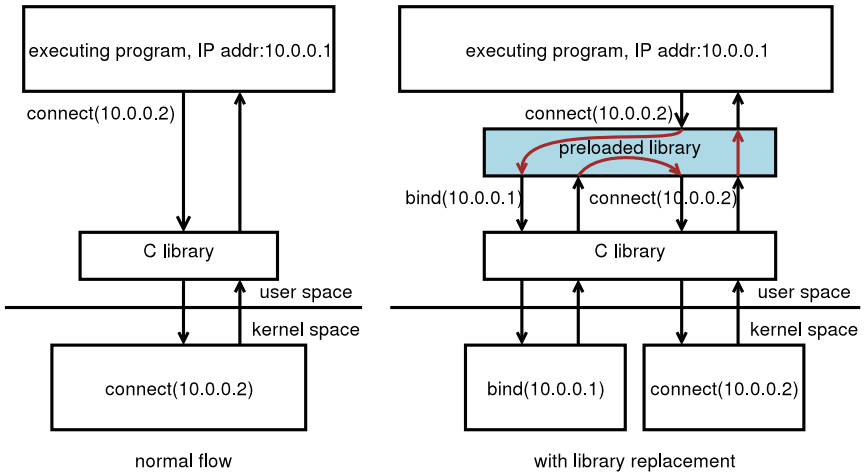

図 3 Flow of execution with and without library interposition

4.4 Features and Limitations

Two basic scenarios can be though of for the deployment of SoftwarePot-based virtual hosting:

- Local hosting
  A site may want to run one or more of its services in confined environments. In this case, it is usually sufficient to install the files on the local file system, and create a Pot file that maps the required files inside the Pot space and executes the service. None of the actual files need to be stored in the archive, reducing the initialization time at Pot execution.
- Remote hosting
  A service provider may provide virtual hosting services based on the presented system. In this case, the deployment of the service can be as simple as sending a single Pot file to the service provider. For users that cannot or do not want to create Pot files for their services, the provider can set up a remote login (ssh) service confined in a Pot environment, in which the user can set up the services to be hosted.

5. Evaluation

In this section we compare our virtual hosting system to some well-known and widely used solutions. The evaluation was performed on two Dell OptiPlex GX260 machines (each with a 2.8GHz Pentium4 CPU, 1 Gb of RAM and Intel PRO/1000 network device), connected through a CentreCOM 9606T Gigabit Ethernet Switch. Benchmarks were performed in the following environments:

- SoftwarePot on Linux 2.4.24
- User Mode Linux 2.4.19 (tt-mode) on Linux 2.4.24
- User Mode Linux 2.4.19 (skas-mode) on Linux 2.4.24
- VMware GSX server
- FreeBSD jail

Unfortunately, benchmark results for VMware GSX server cannot be published due to licence limitations. VMware Workstation 3.2.1 does not have this limitation but it doesn't support our experiment environment. However, according to Barham et al[2], Linux on VMware shows an approximately 30–80% runtime overhead compared to native Linux.

In this test we set up an environment where one machine functioned as a server for different numbers (1, 2, 4, 8, 16, 24, 32) of virtual hosts running Apache HTTPd 1.3.29. The other machine acted as client, utilizing Apache Benchmark (AB) to perform multiple simultaneous requests to the server. Measurements were performed with requests for files of sizes of 10KB, 100KB and 1000KB. One copy of Apache Benchmark was run for each virtual host and all of them were set up to initiate 4 connections at a time (acting as 4 clients). The total throughput of AB clients was measured. Figure 4 shows the results.

Our system provided substantially higher throughput than that of User Mode Linux and often close to that of jail. In the case of 10KB transfers it showed to be scaling significantly better than jail.

It should be noted that when reaching 32 virtual hosts, User Mode Linux tt-mode became highly unstable, so we were not able to perform a valid test during several tries. FreeBSD jail produced some failed requests, making results unreliable. By repeated experiments we could avoid this error in all cases except for the 32 hosts, 1000KB setting. Therefore the result result marked with a star is known not to show the actual throughput of that scenario.

As certain degree of failures is inevitable in highly loaded systems, performing benchmarks for higher numbers of virtual hosts requires reconsidering the examination method, which remains future work.

6. Conclusion and Future Work

We have presented an extension to the SoftwarePot Secure Execution System that makes it capable for utilization for virtual hosting. In the proposed system, services

file size	number of hosts	FreeBSD jail	Linux SoftwarePot	UML skas-mode	UML tt-mode
10KB	01	34249.03	12908.90	5671.76	5586.51
	02	29409.94	12089.30	5760.29	5668.48
	04	22938.74	11082.10	5700.74	5653.48
	08	17448.65	10300.64	5635.02	5503.98
	16	12963.81	8463.53	5375.93	5368.25
	24	7163.91	8093.87	5295.18	5153.04
	32	5570.17	7191.31	4889.95	n/a
100KB	01	72594.62	63773.47	17520.37	17583.48
	02	72627.87	62042.45	17962.07	17589.66
	04	73393.05	58797.29	17784.42	17388.11
	08	72187.86	55353.96	17226.86	17314.51
	16	67604.16	48588.34	16262.19	17090.96
	24	62716.31	46278.31	16336.77	16018.64
	32	53143.01	45298.92	15389.73	n/a
1000KB	01	73037.44	88079.51	23531.56	23115.28
	02	74041.53	87239.96	22942.94	22763.40
	04	73944.19	88851.31	22488.64	23594.20
	08	74976.58	89643.04	22414.98	22176.69
	16	78134.91	90801.36	20956.13	20228.85
	24	79700.09	94788.90	20081.95	20774.74
	32	☆120414.23	94144.51	19639.47	n/a

図 4 Application benchmark – apache throughput

are provided a virtual file system view, allowing them access only to the resources needed. Services can be encapsulated in a single archive file that can be easily deployed to different service providers. In this case. the virtual file system view can absorb the differences between the hosting machines. Virtual hosting of arbitrary services is made possible by forcing all connections by the server program to use the virtual host's IP address.

We performed evaluation benchmarks that show that the proposed system performs significantly better than other user-mode methods and in many cases has a similar performance to kernel-level implementations such as FreeBSD jail. Experiments also showed some weak areas of the system. Most importantly, frequent file accesses cause high runtime overhead for the system as context switches need to be performed for the file path rewriting.

In the future we plan to perform more benchmarks to find out more about which parts need to be improved. Solution should be found to improve the latency of file accesses. Recent versions of Linux and FreeBSD provide directory mapping support at kernel level, these may prove useful in improving the system's performance.

We plan to extend the system to support virtual networking that is suitable different purposes such as testing networked applications. It would be interesting to extend the system to provide framework for testing and development of virtual networking-aware networked software. This should make creating highly connected software such as mobile-agents or Peer-to-Peer programs easier.

参 考 文 献

1) Linux V-Server Project. `http://www.linux-vserver.org/`.
2) Paul Barham, Boris Dragovic, Keir Fraser, Steven Hand, Tim Harris, Alex Ho, and Rolf Neugebauer. Xen and the art of virtualization. In *Proceedings of the 19th ACM Symposium on Operating Systems Principles (SOSP'03)*, pp. 164–177, Bolton Landing, NY, USA, October 2003.
3) Tool Interface Standards (TIS) Committee. *Executable and Linking Format (ELF) Specification*, May 1995. Available from `http://x86.ddj.com/ftp/manuals/tools/elf.pdf`.
4) Jeff Dike. A user-mode port of the linux kernel. In *Proceedings of the USENIX Annual Linux Showcase and Conference, Atlanta, GA*, Oct 2000.
5) Jose Nazario. Using `xinetd`. *Linux Journal*, Vol. 83, pp. 136, 138, 140–141, March 2001. Available from `http://www.linuxjournal.com/article.php?sid=4490`.
6) VMware, Inc. VMware. `http://www.vmware.com/`.