Xenoservers:
Accountable Execution of Untrusted Programs

Dickon Reed, Ian Pratt, Paul Menage, Stephen Early, Neil Stratford *
University of Cambridge Computer Laboratory, Cambridge, UK
Email e.g. Dickon.Reed@cl.cam.ac.uk

Abstract

Many networked applications could benefit from executing closer to the data or services with which they interact. By doing this they may be able to circumvent long communication latencies or avoid transferring data over congested or expensive network links. However, no public infrastructure currently exists that enables this. We propose a system that can execute code supplied by an untrusted user, yet can ***charge*** *this user for all resources consumed by the computation. Such servers could be deployed at strategic locations throughout the Internet, enabling network users such as content providers to distribute components of their applications in a manner that is both efficient and economical.*

We call such a server a ***Xenoserver***[1]*. This paper discusses the construction of such a system, examining how accounting, billing, and quality of service provision can be achieved.*

1. Introduction

It is increasingly recognised that the prevailing model of computation for networked applications is insufficiently flexible. Currently, programmable computation is performed either in client workstations (e.g. web browser) or on remote servers (e.g. web server); in each case the code executing on a system has been trusted not to consume excessive resources.

In some situations either the client or the server might benefit from being able to execute code at, or near, the other end of the connection, for example when submitting multiple inter-dependent queries to a database server. The speed of light places a fundamental lower bound on the latency experienced between endpoints in a network, and hence determines the minimum response time to a query. Similarly there will always be outreaches of the network with little available bandwidth; the optimal place to process a stream of data will often be before it has had to cross congested or low bandwidth links, or links for which a high usage charge is incurred.

At the moment, the owner of a system on the Internet is unlikely to permit the execution of untrusted user-supplied code. This is because existing operating systems are incapable of preventing (possibly unintentional) denial of service attacks due to excessive resource usage by applications. Perhaps the owner may trust Unix or Java Virtual Machines, but these provide only limited security and generally do not address resource management.

Consider an ISP-hosted web server. The ISP owns computing resources at a well-connected location close to the network core and permits customers to store data on the web server for a price. However, few ISPs will allow execution of arbitrary CGI scripts (even though this may be of considerable benefit to the user) for fear of malicious or badly written scripts compromising the stability or performance of the server.

We present an architecture to support organisations in providing computation servers at strategic locations within the Internet that execute untrusted code supplied by third parties. Obviously it is necessary to ensure that user code cannot violate the integrity of the system or interfere with the execution of other (similarly untrusted) applications, so security and stability are concerns. Crucially, *we will bill users for the resources consumed by their applications,* hence financing the service.

1.1. Applications

We perceive many exciting applications for Xenoservers. Web content may be provided by applications running on Xenoservers, for example, by providing programmable front ends to large database servers, or by providing dynamic web content. Rather than relying on web caches, content providers could use Xenoservers to deploy copies of key parts of their web site throughout the network.

*This work was funded by the European Commission ESPRIT Pegasus II Project LTR 21917

[1]From the Greek $\xi\epsilon\nu o\sigma$, a travelling stranger invited into one's house for rest and sustenance.

This would enable them to supply personalised content and maintain accurate access statistics without causing the user to experience poor performance. The provider could arrange for their special web server application to automatically distribute itself around the network to where it is most needed, carrying with it the parts of the web content that are appropriate.

Multimedia stream processing is another potential application; Xenoservers may transcode streams destined for users at the end of congested links, or multiplex and multicast video conferences at the optimal place to reduce latency. Data-mining robots may wander from Xenoserver to Xenoserver, moving closer to databases before communicating with them and then moving on to others.

Xenoservers would be ideal for hosting multi-player games and other virtual reality applications, where reducing interaction latency is critical. A Xenoserver approximately equidistant to all players could be selected whenever a new game session is created. The suppliers of a new multi-player game would not need to invest in a server pool for all their players, instead giving game players the ability to create games on Xenoservers as they choose. The execution charges incurred by the game server could be shared between all the players.

If Xenoservers are to support temporally sensitive applications such as multi-user games and multimedia stream processing then we must provide Quality of Service resource guarantees. The alternative is to substantially over-provision Xenoservers, but operators are unlikely to find this economical.

2. Xenoserver Implementation

Over the last six years we have developed Nemesis [15], a vertically structured operating system for end-user systems. Nemesis has been designed to enable applications to receive quality of service guarantees for all the resources they require, enabling support for high quality multimedia and other soft real-time applications. We are now using Nemesis as the platform for our prototype Xenoserver.

In a Xenoserver all resources that an application consumes — directly or indirectly — should be accounted to the application. Nemesis provides accounting and Quality of Service (QoS) scheduling for most resources. In Nemesis, applications are themselves responsible for performing much of the work of traditional operating system services (using shared libraries). Thus, such work is naturally accounted directly to the applications responsible. (This led to the term "vertical structuring"). Nemesis makes a distinction between control- and data-path operations:

Data-path operations are the frequent, "every day" activities of the computer: delivering and transmitting network packets, file system operations, processing and so forth. Servers and other privileged components only exist on the data-path when they are necessary to provide protection, or control the multiplexing of a resource amongst clients. Our experience with Nemesis has been that such functionality can normally be provided by a 'wafer-thin' privileged kernel layer.

On the control-path — i.e. the rarer connection setup and tear–down, admission control and authorisation events — we use a real-time object request broker. Nemesis uses a single virtual address space with different per-page protection mappings for different trust domains, in order to make inter-process communication fast, yet secure. The control-path servers are carefully designed to perform only small, bounded operations on behalf of clients, so that they do not become a source of Quality of Service crosstalk[2]. Even these small pieces of work can, in some circumstances, be charged back to the client.

Commodity operating systems do not prevent applications attempting denial of service attacks on the operating system by causing shared servers to perform excessive work on their behalf. No matter how good a system's security is, if the system can be made to spend all its time in a high priority server then real applications will make little or no progress. If we are to charge for resource usage meaningfully, we can no longer permit applications a "free ride" from system services.

2.1. Security

Xenoservers will typically run applications provided by many different users. It is essential that these applications cannot interfere with each other. Each application is responsible for its own access control: it can arbitrarily decide to accept or deny requests from other applications. This arrangement is quite different to the one used by Unix and similar systems, where processes sharing a UID can interfere with each other.

It is unsatisfactory for a Xenoserver to need to know in advance the set of users who may run applications on it. If it is open to everybody who can supply digital cash, for example, then the set of possible users is potentially very large. The use of 'accounts', as in Unix and similar systems, is undesirable because it severely reduces the set of machines on which user code may execute.

Applications will typically need to make use of standard system services to allocate memory, set up network connections, obtain processor guarantees, etc. Some system services need to be accessible externally, for example the service that permits new applications to be created. The Xenoserver must ensure that applications (and services)

[2] Quality of service crosstalk occurs where the behaviour of one application leads to the violation of another application's guarantees [17].

cannot interfere with each other, yet provide a mechanism for them to communicate.

In our implementation, access control policy for standard applications and system services is described in a simple language. The system provides a library for interpreting this language, which is based on OASIS [12]. The communication mechanism is an object-based system which securely identifies the origin of messages.

There are many possible policies that applications can use when deciding whether to service requests. Some applications, for example web servers, may choose to service requests without requiring any authorisation. Other applications, like web crawlers and search programs, may only accept requests from their sponsor. More detailed policies are also possible, and applications may choose to make use of the system policy interpreter library.

2.2. Accounting and Charging for Resources

There is no single accounting mechanism suitable for all temporal and spatial resources, so we have to consider each resource individually. All resources are scheduled or allocated, and subsequently charged for. These include:

- CPU cycles spent executing an application.
- CPU scheduling guarantees (whether utilised or not). We currently use the Atropos [17] scheduler.
- Guaranteed response latency to external events (e.g. incoming connections).
- Context switches caused by an application.
- Packets and bytes received and transmitted, including any network usage tolls and possibly device driver and packet filter execution costs. We use the scheme described in [6] to ensure that protocol processing is done in applications, while maintaining network security.
- Disk space rental charges.
- Disk block read and write usage charges and bandwidth guarantees.
- Significant resources used by system servers on behalf of applications while performing control-path operations.

Xenoservers need to provide mechanisms for setting the prices of system resources and for charging users for the total money spent by their applications. Additionally, a mechanism for advertising Xenoserver pricing, availability and network location needs to be implemented, so that potential users can select systems on which to run their applications.

Some owners of Xenoservers can choose to specify the price of resources statically. Occasionally resources might be very cheap or free on some Xenoservers to encourage users onto those machines. Owners may choose to only charge for CPU reservations and response time guarantees and make CPU cycles and network packet reception free. Some Xenoservers could use the free-market resource paradigms described in [7, 18] to set prices.

2.3. System Services

One important system-supplied service will accept programs from users and prepare them for execution. The policy description for this service is likely to vary considerably between Xenoservers. For example, it may refer to online banking services, etc. to check whether the user should be permitted to run a new service.

Nemesis minimises the work done by shared servers. We log the work done by each server on behalf of each application and charge applications for these pieces of work. When transmitting over an Ethernet, for example, a server is necessary to filter packets to ensure they are valid, and this server will also contain a scheduler to perform traffic shaping. We can use the scheduling parameters and logging performed by this server to account and charge for network usage. However, the server does not perform higher level protocol processing, so only a small, constrained amount of work is carried out by the server for each packet. Thus the worst an application can do to the Xenoserver on which it is running is to waste its own money.

If an application wishes to access a protected shared filesystem (rather than a simple flat file), then a server must exist to manipulate the filesystem metadata. Applications are given access to the disk at block level, but the disk device driver filters access according to tables supplied by the filesystem server [4]. This allows work such as the reading of directories, indirect blocks, etc. to be performed by the application, with the filesystem server only being involved in control-path operations such as the opening of files, allocation of new disk blocks, and writing of metadata.

The disk device driver schedules the activity of the disk head itself, and accounts for usage by its various clients. The filesystem server accounts for operations on the metadata. Applications may receive separate resource guarantees for each of these; a service that allows multimedia data files to be accessed over the network may require a large disk bandwidth guarantee but only a very small guarantee from the filesystem server.

It is not essential for a Xenoserver to provide filesystems to its users. Not all applications will require filesystems, and some (for example web servers) may prefer to manage their own set of disk block extents.

2.4. Infrastructure

Users will probably want to limit the money spent by applications (or at least the rate at which money is spent), especially if an application is prone to consuming excessive resources upon failure. The charging mechanism in such cases must have a user-specified upper limit on money spent per application. The billing mechanism should, ideally, not require each user to be registered with each organisation supplying Xenoservers. Ideally, different organisations would agree on a scheme for allowing any customer of any organisation to pay for resources used on any machine.

When a Xenoserver application is started, initially a script is executed. This script typically obtains more code and perhaps some pickled program state, then may use a machine-specific compilation library, or retrieve a cached binary. Xenoserver applications communicate with each other and system services through an ORB. Applications can search for other Xenoservers, checkpoint their state and start new applications remotely. Many common code components such as web server libraries or transcoders will be available that can be shared between applications.

Some applications will need to be written from scratch for Xenoservers. Others may be written in a language such as Java, using a standard Java Virtual Machine shared library. Legacy applications can be ported using shared libraries to emulate existing standards such as POSIX. Newer applications can make use of higher level facilities of Xenoservers to support mobility and persistence. For instance, when a machine becomes loaded, some applications may consult a trader to find a machine where resources are cheaper and move themselves there. A document server could split itself up in to servers running near groups of clients, minimising network charging. Ultimately, with dynamic economic pricing, this may lead to automatic worldwide load balancing, weighted by network performance and charging.

3. Related Work

3.1. Distributed Execution Services

Active Services [2] proposes placing systems in the network that can execute users' applications. This fulfills the goals of allowing program execution to take place closer to data sources. However, the model has no support for resource control or accounting. This is adequate for use within a single organisation, where there is an assumed level of trust between the servers and the clients. To permit the deployment of such systems on the Internet in general, it is necessary to augment the execution model with security and accountability. WebOS [21] provides a similar model with a more developed security mechanism.

3.2. Active Networks

The Active Network [19] community proposes that the network itself — rather than systems within the network — should be made programmable. The user protocol code to be executed at routers in the network may either be supplied with each packet [1, 14] or identified in some way by packet headers and obtained through an out-of-band mechanism [22]. This permits the development of new protocols to be accelerated, since it is no longer necessary for a protocol to be standardised before it can be deployed.

However, available bandwidth — particularly in the network core — is increasing faster than general processing capacity. As a result high performance routers are built with fast forwarding engines in hardware; no processor action is required for the routine forwarding of IP packets. Invoking user-supplied code whenever a packet is received is likely to lead to orders of magnitude lower performance than hardware-based routers.

3.3. Java as a computing platform

The Java Virtual Machine (JVM) is designed to provide a secure, standardised computing platform and as such is an obvious platform to provide on the network. However, the basic JVM provides no facilities for fine-grained resource management. JRes [8] uses binary rewriting techniques to fit resource management on top of a standard JVM. A drawback of this approach is that the JVM specification does not provide the necessary interfaces for measuring resource usage, or for controlling resource allocation and scheduling. An alternative approach [20, 5] is to modify the JVM to support resource control. However, since we do not wish Xenoservers to be restricted to executing only Java code, we prefer to exclude the JVM from the trusted codebase, and provide security and resource management underneath the JVM itself. This ensures that all resources used by Java applications are managed and bugs in the JVM of the kind discussed in [9] cannot compromise the system as a whole.

The JVM becomes just one language interpreter which a user may choose to employ, rather than the computing platform itself — if desired, a user could even make use of a custom JVM. We also do not lose compatibility with the canon of existing code in unsafe languages.

3.4. Resource Control in Operating Systems

Scout [16] associates resources with *paths* (I/O channels spanning multiple processing layers), rather than processes. This permits efficient resource control, but presents problems when running untrusted code since the points for protection boundaries are unclear. Mach [10] and Spring [11] support resource transfer between processes through thread tunnelling. This attempts to solve the problem of QoS crosstalk introduced by use of shared servers on the data path, but with increased complexity when compared with the Nemesis approach of performing work in the client processes. Mungi [13] makes use of an economy-based model for allocation of backing storage, to support garbage collection of unwanted memory segments. In [3], Resource Containers are proposed to allow servers to perform fine-grained resource management between their clients, using hierarchical resource allocations to account for resources consumed within the kernel and shared servers.

4. Conclusion

Xenoservers are machines that can safely and securely perform useful work on behalf of any user who is prepared to pay for the resources consumed. Our experiences with Nemesis [15, 6, 4] lead us to believe that the accounting required can be achieved relatively cheaply. Xenoservers have many useful applications and may revolutionise the way content and services are supplied by allowing application processing to be distributed around the network. We believe that Xenoservers are an essential but currently unavailable component of the computing environment of the future.

References

[1] D. S. Alexander, M. Shaw, S. M. Nettles, and J. M. Smith. Active bridging. *Proceedings of the Conference on Communications Architectures, Protocols and Applications (SIGCOMM)*, Sept. 1997.

[2] E. Amir, S. McCanne, and R. H. Katz. An active service framework and its application to real-time multimedia transcoding. *Proceedings of the Conference on Communications Architectures, Protocols and Applications (SIGCOMM)*, 28(4), Oct. 1998.

[3] G. Banga, P. Druschel, and J. C. Mogul. Resource containers: A new facility for resource management in server systems. In *Proceedings of the 3rd Symposium on Operating Systems Design and Implementation, USENIX Association*, Feb. 1999.

[4] P. Barham. A fresh approach to file system quality of service. *Network and Operating System Support for Distributed Audio and Video (NOSSDAV)*, Apr. 1997.

[5] P. Benradat, D. Lambright, and F. Travostino. Towards a resource-safe Java for service guarantees in uncooperative environments. *Proceedings of the IEEE Workshop on Programming Languages for Real-Time Industrial Applications*, Dec. 1998.

[6] R. Black, P. Barham, A. Donnelly, and N. Stratford. Protocol implementation in a vertically structured operating system. In *IEEE Local Computer Networks '97*, pages 179–188, Minneapolis, Minnesota, Nov. 1997.

[7] S. H. Clearwater. *Market-based control; A paradigm for distributed resource allocation*. World Scientific, 1996.

[8] G. Czajkowski and T. von Eicken. JRes: A resource accounting interface for Java. In *Object-Oriented Programming, Systems, Languages and Applications*, Nov. 1998.

[9] D. Dean, E. W. Felten, D. S. Wallach, and D. Balfanz. Java security: Web browsers and beyond. Technical Report TR-566-97, Princeton University, Computer Science Department, Feb. 1997.

[10] B. Ford and J. Lepreau. Evolving Mach 3.0 to a migrating thread model. In *Proceedings of the Winter 1994 USENIX Technical Conference and Exhibition*, pages 97–114, Jan. 1994. Also Technical Report UUCS-93-022, University of Utah, Department of Computer Science.

[11] G. Hamilton and P. Kougiouris. The Spring nucleus: a microkernel for objects. Technical Report SMLI TR-93-14, Sun Microsystems Laboratories, Apr. 1993. Also in USENIX 93.

[12] R. J. Hayton, J. M. Bacon, and K. Moody. Access control in an open distributed environment. *IEEE Symposium on Security and Privacy*, pages 3–14, May 1998.

[13] G. Heiser, F. Lam, and S. Russel. Resource Management in the Mungi Single-Address-Space Operating System. In *Proceedings of the 21st Australasian Computer Science Conference*, Feb. 1998.

[14] M. Hicks, P. Kakkar, J. T. Moore, C. A. Gunter, and S. Nettles. PLAN: A packet language for active networks. In *Proceedings of the Third ACM SIGPLAN International Conference on Functional Programming Languages*, pages 86–93. ACM, 1998.

[15] I. M. Leslie, D. McAuley, R. Black, T. Roscoe, P. Barham, D. Evers, R. Fairbairns, and E. Hyden. The Design and Implementation of an Operating System to Support Distributed Multimedia Applications. *IEEE Journal on Selected Areas in Communications*, 14(7):1280–1297, Sept. 1996. Article describes state in May 1995.

[16] A. Montz, D. Mosberger, S. O'Malley, L. Peterson, and T. Proebsting. Scout: A communications-oriented operating system. Technical report, Department of Computer Science, University of Arizona, June 1994.

[17] T. Roscoe. The structure of a multi-service operating system. Technical Report 376, Cambridge University Computer Laboratory, Apr. 1995.

[18] N. Stratford and R. Mortier. An economic approach to adaptive resource management. In *Seventh Workshop on Hot Topics in Operating Systems (HotOS-VII)*, Mar. 1999.

[19] D. L. Tennenhouse and D. J. Wetherall. Towards an Active Network Architecture. *Computer Communications Review*, 26(2), Apr. 1996.

[20] P. Tullmann and J. Lepreau. Nested Java processes: OS structure for mobile code. In *Eighth ACM SIGOPS European Workshop*, Sept. 1998.

[21] A. Vahdat, T. Anderson, M. Dahlin, E. Belani, D. Culler, P. Eastham, and C. Yoshikawa. WebOS: Operating system services for wide area applications. In *Proceedings of the Seventh Symposium on High Performance Distributed Computing*, July 1998. `http://www.cs.utexas.EDU/users/dahlin/papers/hpdc98.ps`.

[22] D. J. Wetherall, J. Guttag, and D. L. Tennenhouse. ANTS: A toolkit for building and dynamically deploying network protocols. *Proceedings of IEEE OPENARCH '98*, Apr. 1998.