The Xenoserver Computing Infrastructure

A project overview

K A Fraser, S M Hand, T L Harris, I M Leslie & I A Pratt

Abstract

The Xenoserver project [15] will build a public infrastructure for wide-area distributed computing. We envisage a world in which Xenoserver execution platforms will be scattered across the globe and available for any member of the public to submit code for execution. Crucially, the code's sponsor will be billed for all the resources used or reserved during its execution. This will encourage load balancing, limit congestion, and make the platform self-financing.

Such a global infrastructure is essential to address the fundamental problem of communication latency. By enabling principals to run programs at points throughout the network they can ensure that their code executes close to the entities with which it interacts. As well as reducing latency this can be used to avoid network bottlenecks, to reduce long-haul network charges and to provide a network presence for transiently-connected mobile devices.

This project will build and deploy a global Xenoserver test-bed and make it available to authenticated external users; initially members of the scientific community and ultimately of the general public. In this environment accurate resource accounting and pricing is critical – whether in an actual currency or one that is fictitious. As with our existing work on OS resource management, pricing provides the feedback necessary for applications that can adapt, and prevents over-use by applications that cannot.

1 Introduction

Technologists often envision a future with distributed services acted on by mobile agents that traverse the network at a user's behest, for example to be close to data sources, to co-locate for interaction or simply to migrate away from congestion. Unfortunately, and despite existing work on agent technology and network programmability, this tantalizing vision will remain just that until a number of key research challenges are addressed.

These challenges are (*i*) the need for a *public* and *ubiquitous* infrastructure that can be used (with suitable control and audit facilities) over timescales of the order of minutes or hours rather than just months, (*ii*) the need for a *flexible* system to support code written for a variety of languages and paradigms and (*iii*) the need to support interaction and sharing in a *widely distributed* environment with communication subject to delay and clients subject to disconnection.

For example, consider the case of a number of players wishing to participate in an on-line game or other virtual reality situation. The Xenoserver platform would enable them to run their game server on on a machine at a point in the network such that the maximum round trip time between the server and any of the players is minimized. They would be able to rent a "slice" of a machine sufficient for their purposes for the duration of their game, perhaps just a few minutes.

Similarly, the platform could be used to host distributed services, perhaps a web site with dynamically generated content, a streaming media distribution system. The service administrator (or agent acting on their behalf) could arrange to replicate the service according to demand by buying time on Xenoserver machines. They could even arrange that service replicas are distributed according to access patterns and thus exploit locality [24].

Xenoservers are an ideal substrate for hosting web service components such as those proposed in the Open Grid Services Architecture [10], or the platforms contemplated by the EU Global Computing initiative [1]. They can provide a realisation of the resource management functions proposed in the Grid Resource Allocation and Management architecture (GRAM), and enable the Grid to extend out onto "public" servers. Other web services related technologies such as UDDI and WSDL provide the infrastructure required to enable clients to locate transient components being hosted on Xenoservers enabling services to be relocated far more readily than is possible with current DNS based techniques.

Xenoservers expose a very flexible execution interface, implemented at a low level by a simple hypervisor, running in place of a full-featured operating system. This securely partitions the services it hosts, and accurately accounts the resources that each consumes. This low-level approach enables Xenoservers to support a wide range of services. For example, users could each run their own operating system above the hypervisor. A user may wish to rent a partition of a server on which to run a particular version of Linux, upon which they will run their own application (perhaps an experimental mobile agent platform, an active network implementation, or a server for a collaborative virtual environment). Resource management and QoS provisioning will support multiple concurrent tasks. In some cases, users will design applications to run directly on the hypervisor for improved efficiency. Our experience with hosting the JVM over Nemesis suggests that useful optimizations can be made – for example to share the results of run-time compilation, or to control how the system pages parts of a garbage-collected heap. Experience with developing the Nemesis operating system has shown us that a further benefit of the low-level interface approach is that resource accounting and QoS firewalling is made easier.

Key to the Xenoserver architecture is "Xenocorp" which acts as a broker and introduction service between Xenoservers and clients; it authenticates clients, matches client requirements with server availability and maintains billing information. Although the interaction between a client and a particular Xenoserver may be transient, both the client and the Xenoserver's owner have a longer term contractual relationship with Xenocorp. Between them Xenocorp and Xenoserver owners maintain an audit trail of the network activity of applications running on the platform. This can be used (under warrant) to identify the sponsor of an application later identified to be behaving illegally.

Note that Xenocorp does not represent a centralized single point of failure: most of its functions may be achieved by renting time on Xenoservers – either from the public infrastructure or, for trusted services, on dedicated Xenoservers. There could even be multiple Xenocorp instances, with Xenoservers' owners registering their machines with more than one organisation. The organisations would compete on the basis of their ability to meet users' requirements, much like WWW portals do today.

Xenocorp will maintain knowledge of the network topology around each Xenoserver, enabling it to identify servers that best match clients' communication latency requirements. Since the price that individual Xenoservers charge for resources will fluctuate according to the demand they are currently experiencing Xenocorp will maintain a database of these charges and use this information to match any cost constraints described in the clients' requests. Middleware will be developed to assist client applications in determining their resource requirements and making decisions about how to react to changes in resource price, etc.

1.1 Goals

In summary, our main goals are to:

- develop an efficient Xenoserver platform for hosting services and managing the resources they use,
- investigate techniques for automatic service placement and adaptive resource management in a wide-area setting with a diverse mix of clients,
- develop a scalable system for managing access control and auditing in this distributed setting,
- provide effective programming aids and services for developing code to operate on Xenoservers, in particular in combination with mobile devices or widely distributed clients,
- as a demonstration of this project (and as a tool for others), use Xenoservers to deploy one or more existing active network systems developed under UK research programmes.

2 Research context

In this section we briefly survey research in service deployment, system virtualization and pervasive computing.

Deployment systems. Existing service-hosting companies rent access to Internet-connected servers over reasonably long timescales – booking involves personal interaction with the company and minimum periods of several months are typical. This fits with the operators' expected market of hosting long-running services, such as web sites or off-site storage facilties. Only recently have operators begun to consider the possibility of hosting smaller services over shorter timescales [16].

The success of existing infrastructure providers such as Akamai and Digital Island indicates that there is demand for third-party service distribution within the Internet. These existing infrastructure providers host replicas of client web sites and use DNS redirection to route service requests to the current optimal replica in each section of the Internet.

Within computational grid projects, the Globus software, provides facilities for remote job execution and security management within a heterogeneous network. The current version is described as not offering 'substantially more functionality than a set of scripts and remote shells' [3] for managing jobs on remote high-performance machines drawn from a federated user community.

Virtual machines. Virtual machines are partitions of complete systems that provide an execution environment to the tasks running within them. This virtualized environment may be the same architecture as the host machine (e.g. providing a mechanism to run multiple OSs on a single computer) or it may be radically different (for example the Java Virtual Machine which provides its own instruction formats and extensive libraries).

Virtualization is prevalent on large server systems (for example *logical partitions* within the IBM S/390 and zSeries architectures, Sun *dynamic system domains* or *virtual partitions* on HP Superdome systems). Each of these systems provides the ability to run several isolated OS instances over a single physical machine by dividing the CPUs, I/O devices and memory between the various domains.

Virtualization has also been explored on desktop systems, although in a quite different mode to those above: the base system is being multiplexed between the environments that it supports, often under the control of a single user, rather than being partitioned into them for use by separate users or groups. The reallocation of resources to domains therefore occurs over short timescales (milliseconds) to support execution, rather than over longer timescales to adapt to shifting load. Environments such as the Java Virtual Machine (JVM) and Microsoft CLR have gained currency because they provide an OS-independent interface to applications. Similarly, on workstations, systems to run separate OS instances (such as VMWare) are used for software compatibility.

Resource management and task isolation has been the focus of ongoing work since the original deployment of the JVM. The state of the art is to use support within the JVM to track resource usage and provide isolation within tasks [6]. Czajkowski *et al* suggest that this is the first scheme that is suitable for use in earnest: earlier developments placed significant restrictions on application behaviour (for example, forbidding the usual *reflection* API) or imposed further run-time checks on frequent operations.

Pervasive computing. The development of agent systems has been a popular topic of research into distributed systems – [23] provides a contemporary introduction. Such systems may be seen as precursors to general programmable networks in that they host remotely-supplied code, often so that operations can be performed close to the data with which the code is to interact.

The need for inter-operability between agent systems has recently been recognised. The OMG, maintainers of the vendor-neutral CORBA standard for distributed objects, attempted to define a common API which could be supported by existing systems [13]. They define a common terminology (*agents*, *authorities*, *places* and the like) and general operations to transfer, to name and to locate agents. However, while aiming for portable implementation, it assumes the ubiquity of the Java programming language and does not aim to provide a general platform for remote or mobile execution.

This *write-once, move anywhere* goal has also been proposed by Grimstrup *et al* [7]. They again address interoperability between the APIs of different agent systems, defining translations to and from existing APIs and a common format. They also assume that Java will be used throughout.

Several ongoing projects are investigating general programmable networks. SafetyNet at Sussex University expresses network security policies by extending the type system of an active programming language [11]. Lancaster Active Node (LANode) focusses on per-packet processing and presents two switch interfaces to the programmer: a Java-based control interface, and an efficient switch-specific data-forwarding interface (such as LARA NodeOS [5]). The UCL XMILE project has developed an XML-based programming language supporting fine-grained code mobility between active switches. These projects build on existing work such as the MIT Active Node Transfer System (ANTS) [22], and University of Pennsylvania's SwitchWare architecture [2]. In addition, there is ongoing work to develop general OS support for *liquid software* [9, 19].

One application of Xenoservers is to host a test deployment of these existing mobile agent platforms at strategic places within the Internet, providing a common substrate for the authentication and resource management that they require for public use.

Summary. Existing work provides piece-wise solutions that cannot be combined cohesively. Firstly, service hosting schemes have a high entry cost and are designed for use with manual and contractual intervention. They are unsuitable for fast deployment of ad-hoc or experimental. Secondly, facilities for partitioning moderately sized machines are geared for software compatibility. We require a system with accurate resource accounting at the level of individual partitions – perhaps hosting several dozen on an ordinary workstation. Finally, mobile agent systems, which present a natural mechanism for deploying client software in a ubiquitous network, have not gained currency in a public environment. Existing systems do not integrate resource management, audit facilities, operation on behalf of public users or cross-language support.

3 Work programme

We describe our proposed work in three packages, essentially corresponding to three concurrent strands of activity. The first, described in Section 3.1 is the server platform operating on each Xenoserver, hosting applications, services or agents. The second, described in Section 3.2 is the management infrastructure which we term *Xenocorp*. The third, covered in Section 3.3 are facilities to aid the operation of code on Xenoservers, supporting *latency-tolerant distributed computing*. Finally, Section 3.4 describes our global deployment strategy.

3.1 Server platform

Each Xenoserver acts as a host for the applications, services or agents deployed on it by clients and authenticated by Xenocorp. The platform must provide isolation between these clients, both in terms of access control (to allow safe execution of mutually untrusting code on the same hardware) and in terms of resource consumption (so that clients may be charged according to their usage).

W1.1 Core platform development. The core platform requirements of resource accounting and isolation fit closely with our experience developing the Nemesis operating system [12, 8]. Our basic platform design will follow a similar model: a low-level *hypervisor* provides protection, scheduling and accounting. However, rather than running applications directly over this (as we did in Nemesis) we intend to host traditional OSs – these form the unit of resource management, rather than individual applications. Applications with soft real-time QoS processing constraints can purchase fine-grained resource reservations.

A client of the Xenoserver may elect whether to instantiate its own OS instance (guaranteeing isolation, but at a higher resource cost), to deploy its code over an existing instance (with weaker isolation, but lower start-up costs) or to use a specialized resource-managed platform, such as our existing RCANE active-networking environment (providing isolation, but requiring specialized client code).

W1.2 Guest OS development. The interface between the core platform and guest OS will be designed with low-cost virtualization in mind, rather than complete transparency. For example, unlike VMWare, we do not intend to host unmodified operating systems by emulating in software processor features that are difficult to virtualize. Our experience with Nemesis informs us that by sacrificing complete transparency the cost of virtualization can be reduced: we aim for the hypervisor to be able to support a substantial number of concurrent instances of guest OSs, rather than the small number that may be managed by VMWare. The cost to the hypervisor of running an instance of a suitably-developed guest OS should be comparable to the cost of an ordinary system running a process.

We will take Linux as our first guest OS and follow a simple incremental approach: firstly defining special interfaces, exported by the core platform, for each privileged instruction required by the Linux kernel and then replacing these with efficient compound operations – for example propagating page-table updates proposed by the guest OS in batches. We can take a similar approach to device virtualization, introducing sharing on a per-device basis.

Linux provides a familiar POSIX environment and many existing server-type applications and demonstrators. They can thus run unmodified on our platform. Linux also serves as a host for other APIs, such as the JVM, Microsoft CLR, the Globus Grid toolkit, and active network node implementations such as the LARA NodeOS.

APIs will be added to our port of Linux to enable applications to control the resources allocated by the hypervisor and receive information about current resource prices, athough reasonable default behaviour (based on the sponsor's initial specification) will be provided so that applications may run unmodified.

W1.3 Resource management. Although the mechanisms for resource management in the core platform resemble those used in Nemesis, they differ fundamentally in the timescales over which changes are made and in the models of allocation and revocation. As a workstation OS, Nemesis assumes that all applications are controlled by a single user who acts as an ultimate arbiter: the available resources are fixed and the user's own goals guide the allocation policy.

On a Xenoserver we rely on economic feedback to users to control application resource consumption. In this work item we will investigate such mechanisms for *distributed* resource management, building on our experience of economic approaches to management of a single machine [21, 14]. The distributed setting raises challenges in terms of how frequently the entities involved communicate (short timescales allow rapid adaptation to varying resource availability, but load the participants and intervening network) and what occurs when communication fails. One promising approach is for such policies and trade-offs to be under the control of a client-supplied agent, also hosted on the Xenoserver, to which the client can delegate control over short timescales but which requires re-authorization beyond a certain resource budget. The interfaces available to such an agent, the language in which it is expressed and the accounting of its own resource usage are all areas for study.

3.2 Management infrastructure

Xenocorp acts as a broker and introduction service between clients and Xenoservers; it authenticates clients, matches client requirements with server availability and maintains billing information. Although the interaction between a client and a Xenoserver may be transient, both would have a longer term contractual relationship with Xenocorp.

W2.1 Base management platform development. This work item will develop the basic management functionality that is required, namely the authentication of prospective clients, storage of client account information, and maintenance of basic directory information about available Xenoservers. Items W2.2–W2.4 develop this further.

W2.2 Resource discovery. We envisage client code being supplied for execution for a variety of purposes, for example: an organization may want to deploy a prototype service, or small-scale service; a group of users may want to deploy a server for their own private use (perhaps to control a multi-user game or as storage for collaborative work); or an individual user's application may wish to execute a number of short running queries close to some data source. In each of these cases the code being executed has distinct positioning requirements in terms of latency and bandwidth to various network endpoints, and of course the users may have different levels of funding for their endeavours.

Different applications may also request different execution environments, not all of which can be efficiently supported by all Xenoservers. For example, some programs may have quite explicit requirements such as "Redhat Linux 7.2 on Intel x86", whereas a program requiring "Java 1.4" could probably have it's necessary environment assembled on most CPU architectures. The management infrastructure is responsible for matching all these requirements with the facilities available from the Xenoservers under its control.

In this work item we will investigate higher level questions of resource allocation during client-code deployment, as distinct to the adaptation and control interfaces that are used once a job has been sited on a Xenoserver (W1.3). Key questions are how to express requirements in a flexible manner and how to enable efficient matching. The use of a hierarchical XML format for expressing requirements and an extended tree matching language is one concrete option.

W2.3 Network audit-trail management. The prospect of a publically-available computing platform raises obvious concerns over its potential for nefarious use – in particular in its rôle as another level of indirection between a user's own machine and a target that they may wish to attack over the network. Taking this as the major risk for misuse of the system, the Xenoserver infrastructure as a whole should be able to identify the user initiating a particular network connection from a Xenoserver. However, rather than logging information about all connections we can extend the rôle of the hypervisor since that is required, in any case, to ensure that packets being sent by a guest operating system are correctly formed with an appropriate source address and protocol settings.

We will evaluate a range of schemes, in terms of the resource budget of the client, the run-time overhead and the kinds of network access that clients are permitted. At one extreme a globally-unique IP address, owned by the local Xenoserver, can be leased to the client, and serve to identify it – this scheme is easy to enforce by packet filtering and information need only be logged on address re-use. At the other extreme the hypervisor may write fingerprints into packets to identify their provenance – clients may share a single IP address, but there is a per-packet overhead of creating or checking fingerprints. Song and Perrig's scheme illustrates this form of fingerprinting [20]. Furthermore, the hypervisor could prevent many common types of Denial of Service attack by checking that network flows exhibit reasonable packet rates in both directions; a large excess of outgoing packets indicates a client may be behaving suspicously.

W2.4 Recursive implementation. A running theme through the development of the base platform, management infrastructure and service facilities is the tension between flexibility afforded to 'control path' operations such as resource negotiation and the ability to account the resources that such operations use. For example, a candidate for both W1.3 and W2.2 would be to use arbitrary code to express client policies and preferences (much as the Nemesis OS does).

Throughout the project we will investigate how much of the functionality provided needs to be considered an inviolable part of the Xenoserver infrastructure and which can be executed over it (subject, therefore, to resource accounting). We suspect that much of the management infrastructure can be provided in this way, with only the most basic directory information maintained within the core. The problems again resemble those of OS design but in a distributed setting and with a need to consider latency and failures.

3.3 Service facilities

We envisage that many of the applications hosted by Xenoservers will be written in a conventional style, using existing development tools and libraries. Under the *Service facilities* work package we propose developing techniques that can be used to aid efficient execution in the face of comparatively long communication delays and failures.

W3.1 Storage management. Having access to a shared distributed file system will assist users in efficiently moving their code and data between Xenoservers. Code components used by applications will often be common between multiple users. For example, the guest OS executables and components such as JVMs and web servers can be digitally signed by their author and then can be cached and used by anyone who trusts the originator. Other application components, and data files in particular, are likely to be private to particular users. The shared file system provides a convenient and relatively efficient way of allowing users a uniform environment across all Xenoservers.

The Grid Storage Resource Broker [4] offers some of these facilities, but does not yet address the quota management and automatic caching and replication required in a Xenoserver environment. We have begun the design of a system called *Pasta* which makes use of the Pastry [17] peer-to-peer routeing substrate. Unlike most other peer-to-peer file systems, Pasta explicitly addresses issues such as file mutability and permanence. It will make use of the Xenoserver billing mechanisms to charge users for the data they store, and to pass on charges incurred during retreival.

Pasta has a very flexible and extensible scheme for naming in a decentralised system. Users can construct their own private file system name spaces, and then selectively share them with other users. As well as directories and files, user name spaces can 'mount' other exported name spaces. Overlay and union mounts enable functions akin to copy-on-write etc. Since Pasta is *versioning* file system, it is possible for users to select specific archived versions of files or to access the most recent. Users can keep any file in the system permanent by agreeing to share the cost of its storage.

W3.2 Latency tolerant protocols. Existing protocols for communicating and sharing data are usually based on caching and multiple-reader-single-writer ownership (for example NFS v4 or CODA [18]). Lock ownership is particularly problematic in the presence of failures since system-wide liveness requires the continued operation of the lock holder. Our ongoing work on lock-free data structures has developed practical techniques for coping with delayed tasks or system failures in shared-memory multiprocessor systems. We will investigate how similar techniques can be used here, accessing remote shared data and meta-data through atomic primitives.

As with the atomic operations available on microprocessors, a small set of primitives can be used both to express existing protocols (enabling un-modified client APIs to be used) and also to allow specialised new designs. Providing *load-linked* (LL) and *store-conditional* (SC) operations acting at a byte-range level are promising candidates for inclusion as basic operations – these can be used to build ordinary locks for use by existing APIs and also to express bespoke lock-free communication services.

W3.3 Virtualized debugging. When debugging code, distribution poses even more substantial problems than those of single-process systems: separate debuggers must be attached to the various processes involved. There is no central way to control system-wide parameters – perhaps to impose particular loss patterns or delays on communications, to corrupt messages or to control the relative execution speeds of different components. Such features must either be intrinsic to the platform running the tests or must be implemented as additional testing code by the programmer.

To address these problems we propose the technique of *virtualized debugging* in which all of the resources used by the system under test are virtualized by the debugger – ranging from low-level details such as the precise implementation of processor instructions to the scheduling of threads, the provision of separate virtual address spaces to different processes and network communication between those processes. Retaining control over the resources in use allows the debugger to ensure deterministic execution, to expand or contract the detail with which parts of the system are modelled, to manage distributed applications as single entities and to control the performance of external components such as communication links.

The Xenoserver architecture provides an ideal setting in which to evaluate this approach.

3.4 Deployment and experiementation

In the second and third years of the project we plan to deploy at least ten prototype Xenoservers in well-connected network locations around the world.

Some of the test nodes will come through re-deployment of the initial development platforms among collaborators, others will be procured by renting "dedicated servers" from specialist hosting companies. Rather than using a single international hosting company to provide machines in a number of locations, it is more cost-effective to deal with small companies in individual cities. This will have the additional benefit of resulting in a more rich and representative network topology.

After an initial private testing phase, we plan to make the platform available to other research groups wishing to trial distributed applications and services. We have already been in contact with groups working on potential candidates for deployment, including active network nodes, wide-area event distribution systems, and Internet topology discovery and measurement tools.

Our ultimate goal is to make the trial system publicly available, where users register via a web site and receive some Xenoserver currency to spend as they desire. At this point an economic analysis of the system can be conducted to discover how the resource pricing algorithms work in practise, and how they might be improved to achieve the desired goals of load balancing and congestion control while maintaining stability. This would also be an opportunity to assess the security and auditing capabilities of the platform.

A further topic of investigation is to determine whether such a platform is realistically self-funding, or perhaps even profitable. We envisage a scenario whereby the actual Xenoserver machines are owned by a wide range of companies such as current server hosting facilities, ISPs, or even service companies such as Altavista or Google that host databases that Xenoserver users may wish to access. The cost of running the Xenoservers would hopefully be recouped through the charges incurred by users and distributed to the owners through Xenocorp. Xenocorp would in turn pay charges to the various Xenoserver owners for hosting components of the management infrastructure it runs. Xenocorp itself might be funded through taking a percentage of each transaction.

It is only through having a deployed prototype with some example applications that we can discover whether there would be the user demand and willingness to pay to make the public infrastructure we propose viable.

4 Beneficiaries, dissemination

Within the short term the direct beneficiaries of this work will be the research groups that will be able to use Xenoservers as an experimental platform for wide-area evaluation. The benefits of doing so will be that system behaviour can be explored more thoroughly than (for example) through simulation or through deployment over local workstation clusters. Agent technology and active networking is a thriving area within UK academic research, but currently lacks the practical grounding and global-scale deployment that Xenoserver technology can bring.

Over the longer term, Xenoservers stand to have a pervasive influence on deploying and managing software. A platform of the kind we intend to develop – integrating flexibility, authentication and resource management – promises to be a key part of next-generation network services. It will act both as a host for code from poorly-resourced mobile devices and as a mechanism for exploratory or rapidly-adaptive service deployment.

During the course of the project we intend to disseminate the key academic results through meetings and through publication. We will hold twice-yearly open meetings at which interested parties will be invited to attend presentations (at their own expense) on the progress of the work and contribute to discussions about direction. We envisage each of the items of work identified in Sections 3.1–3.3 being accompanied by a report suitable individually for workshop presentation and ultimately as substantial components of PhD dissertations and full conference papers. We intend to produce at least one such major publication combining the experiences from each of the work packages, targeting SOSP, OSDI, PODC or PLDI. Our final report will draw together that work and incorporate experiences from early adopters of the platform.

5 Success criteria

Beyond the goals of Section 1.1, in terms of resource management, scalability and application support, we wish to demonstrate the success of this project by it (*i*) providing clear pointers to new research areas enabled by pervasive global computing and (*ii*) developing a concrete model and experimental platform for economic-based resource management in a wide-area distributed setting.

Corresponding to each goal, we have the following acceptance criteria:

- The overheads introduced by executing on the Xenoserver platform should be low: the resources needed to host an application on a Xenoserver should be comparable to those needed to host it over an ordinary OS.
- Xenocorp should provide sufficient services that user authentication and service placement can be automated to a degree acceptable to a casual user.
- Our final demonstrations should show how the system will scale to support large numbers of users (1000s) and large numbers of servers (100s), ideally by using the resulting Xenoserver to host many aspects of the management functionality.
- We should be able to demonstrate clear benefits by the new techniques for storage management and communication that we will investigate – ideally with examples of these techniques having been found effective by early adopters of the platform.
- Our resulting infrastructure should be open for use by other programmable networks research groups. Demonstration applications should show how the system can be used to support ubiquitous computing.

References

[1] Global computing, FET proactive initiative, 2001, `http://www.cordis.lu/ist/fetgc-sy.htm`.

[2] ALEXANDER, D. S., ARBAUGH, W. A., KEROMYTIS, A. D., AND SMITH, J. M. A secure active network architecture: Realization in SwitchWare. *IEEE Network 12*, 3 (1998), 37–45. Special Issue on Active and Controllable Networks.

[3] ALLAN, R. J., BOYS, D. R. S., FOLKES, T., GREENOUGH, C., HANLON, D., MIDDLETON, R. P., AND SANSUM, R. A. Globus and associated Grid middleware. consolidated evaluation report from UKHEC sites. Tech. rep., UKHEC, 2001.

[4] BARU, C., MOORE, R., RAJASEKAR, A., AND WAN, M. The sdsc storage resource broker, 1998.

[5] CARDOE, R., FINNEY, J., SCOTT, A. C., AND SHEPHERD, W. D. LARA: a prototype system for supporting high performance active networking. In *Proceedings of the First International Working Conference on Active Networks – IWAN '99* (June 1999).

[6] CZAJKOWSKI, G., AND DAYNÈS, L. Multitasking without compromise: a virtual machine evolution. In *OOPSLA 2001*.

[7] GRIMSTRUP, A., GRAY, R. S., KOTZ, D., COWIN, T., HILL, G., SURI, N., CHACON, D., AND HOFMANN, M. Write Once, Move Anywhere: Toward Dynamic Interoperability of Mobile Agent Systems. Tech. Rep. TR2001-411, Dartmouth College, Computer Science, Hanover, NH, July 2001.

[8] HAND, S. M. Self-paging in the nemesis operating system. In *Proceedings of the 3rd Symposium on Operating Systems Design and Implementation (OSDI '99)* (Feb. 1999), The USENIX Association, pp. 73–86.

[9] HARTMAN, J., MANBER, U., PETERSON, L., AND PROEBSTING, T. Liquid software: A new paradigm for networked systems. Tech. Rep. TR96-11, Department of Computer Science, University of Arizona, 1996.

[10] I. FOSTER, C. KESSELMAN, J. N., AND TUECKE, S. The physiology of the grid: An open grid services architecture for distributed systems integration., January 2002.

[11] JEFFREY, A., AND WAKEMAN, I. A survey of semantic techniques for active networks. Available from the SafetyNet Project web site, `http://www.cogs.susx.ac.uk/projects/safetynet/`.

[12] LESLIE, I. M., MCAULEY, D., BLACK, R., ROSCOE, T., BARHAM, P., EVERS, D., FAIRBAIRNS, R., AND HYDEN, E. The design and implementation of an operating system to support distributed multimedia applications. *IEEE Journal on Selected Areas In Communications 14*, 7 (Sept. 1996), 1280–1297.

[13] MILOJICIC, D., BREUGST, M., BUSSE, I., CAMPBELL, J., COVACI, S., FRIEDMAN, B., KOSAKA, K., LANGE, D., ONO, K., OSHIMA, M., THAM, C., VIRDHAGRISWARAN, S., AND WHITE, J. MASIF: The OMG Mobile Agent System Interoperability Facility. In *Proceedings of the 2nd International Workshop on Mobile Agents* (1998), K. Rothermel and F. Hohl, Eds., vol. 1477 of *Lecture Notes in Computer Science*, Springer-Verlag: Heidelberg, Germany, pp. 50–67.

[14] Oparah, D. *Adaptive Resource management in a Multimedia Operating System*. PhD thesis, University of Cambridge Computer Laboratory, 2000.

[15] Reed, D., Pratt, I., Menage, P., Early, S., and Stratford, N. Xenoservers: accounted execution of untrusted code. In *Proceedings of the fifth Workshop on Hot Topics in Operating Systems (HotOS-VII)* (1999).

[16] Roscoe, T., and Lyles, B. Distributing Computing without DPEs: Design Considerations for Public Computing Platforms. In *9th ACM SIGOPS European Workshop, Kolding, Denmark* (September 2000).

[17] Rowstron, A., and Druschel, P. Pastry: Scalable, decentralized object location, and routing for large-scale peer-to-peer systems. *Lecture Notes in Computer Science 2218* (2001), 329–??

[18] Satyanarayanan, M. Fundamental challenges in mobile computing. In *Fifteenth ACM Symposium on Principles of Distributed Computing* (Philadelphia, PA, May 1996). http://www.cs.cmu.edu/afs/cs/project/coda/Web/coda.html.

[19] Shapiro, J. S. Operating system requirements for liquid software. Tech. Rep. SRL-2000-02, Department of Computer Science, Johns Hopkins University, 2000.

[20] Song, D. X., and Perrig, A. Advanced and authenticated marking schemes for IP traceback. In *Proceedings IEEE Infocomm 2001* (Apr. 2001).

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

[22] Wetherall, D., Guttag, J., and Tennenhouse, D. ANTS: Network services without the red tape. *Computer 32*, 4 (Apr. 1999), 42–48.

[23] Wong, D., Paciorek, N., and Moore, D. Java-based Mobile Agents. *Communications of the ACM 42*, 3 (Mar. 1999), 92–102.

[24] Yan, J., Early, S., and Anderson, R. The XenoService – a distributed defeat for distributed denial of service. In *Proceedings of the Information Survivability Workshop (ISW)* (Oct. 2000).