Distributed resource discovery and management in the XenoServers platform

Evangelos Kotsovinos and Timothy L Harris
University of Cambridge Computer Laboratory

Contact: E Kotsovinos, Computer Laboratory
J J Thomson Avenue, Cambridge, UK, CB3 0FD
Evangelos.Kotsovinos@cl.cam.ac.uk

August 2002

Abstract

In this paper we present the main ideas behind the design of the XenoServers distributed platform, which substantiates a public infrastructure for wide-area distributed computing. We present our initial design of the distributed architecture, and emphasize on ways of locating and administering distributed resources in this large-scale, federated platform. The XenoServers global infrastructure is essential to address several fundamental problems of today, such as communication latency, network bottlenecks and long-haul network charges.

1 Introduction

The XenoServers [11, 4] project aims to build a public infrastructure for wide-area distributed computing. Clients from any place in the world will be able to submit code for execution on our platform, and the sponsor of the code will be billed for all the resources used or reserved during the course of execution.

The XenoServers architecture is essential to help addressing some significant problems and needs of today. First, as studies [10, 1, 9] show, unlike previously prevailing opinion [2] there appears to be a strong link between geographic distance and communication latency. The XenoServers platform supports the placement of servers physically close to the clients, and therefore helps *reducing latency and network load*. Also, architectures like *computational grids* [8] or *distributed multimedia servers* [3] can find an ideal testbed environment on our platform, having the opportunity to be deployed easily and efficiently on a large number of XenoServers around the world. Additionally, our platform is a perfect infrastructure for deploying *mobile agent-based applications*, as XenoServers can host the execution platforms required by agents and allow agents to migrate between participating XenoServers. Likewise, *mirroring web services* for increasing content availability and balancing load will become a lot easier, dynamic and feasible in short timescales with the use of XenoServers.

XenoServers are fundamentally different from Computational grids, as the latter denote virtual supercomputers, constructed dynamically from geographically dispersed and heterogenous resources, linked by high-speed networks to complete some large-scale work. Those architectures provide application-level programming models and interfaces for resource sharing rather than system-level support. Resource management in such systems is focused on grouping and mapping distributed resources to local ones, rather than administering and sharing distributed resources between competing users. Metacomputing infrastructures would find XenoServers a highly suitable platform to run on.

This paper is focusing on the higher level design of the XenoServers platform and the proposed resource discovery and management mechanisms, whereas [5] discussed the design of the Hypervisor module, briefly described later in Section 2. Also, it has to be pointed that there are several issues on areas like security and auditing, that are an integral part of the system, but are not covered in detail in this paper. The structure is as follows. Section 2 outlines the design of our distributed architecture, Section 3 discusses the resource discovery and management issues, and Section 4 summarizes our conclusions.

2 XenoPlatform Design

The main entities of our distributed platform are clients, XenoCorp servers, and XenoServers. *XenoServers* are hosted by remote machines around the Internet, and it is there that jobs are dispatched for execution. In order to make this possible, a part of the XenoServers software is running in a layer called the *Hypervisor* [5] between the hardware and the *execution contexts*, which the system hosts. These contexts may include versions of traditional operating systems ported to run over the Hypervisor –we have already developed a Linux context – or specialized environments, such as a resource managed Java Virtual Machine, which can run without an intervening operating system. Having a part of XenoServers running in such a low level, allows for accurate resource isolation and accounting to be performed in a per-execution context basis.

XenoCorp servers support the functionality of our system, operating as a distributed directory for keeping track of active XenoServers and registered clients. Clients have to register with XenoCorp before they start using our platform. One of the core functions of XenoCorp is to supply clients with information about available XenoServers and their specifications, which means that XenoCorp undertakes the process of finding an appropriate XenoServer on behalf of the clients. Also, XenoCorp has to handle the identification, authentication and charging of clients. XenoCorp servers and XenoServers have to be in regular communication. XenoCorp needs to be informed by the XenoServers about the resource usage done by each client, in order to charge clients. Also, XenoServers have to be able to check whether a client that has submitted a job is registered and authorized to use the system.

It would be inappropriate to fix policies on issues like pricing and charging at the system design stage –it is possible to conceive of many different models which are appropriate to different circumstances. In order to support both choice and diversity, we believe that the XenoPlatform should be able to support *multiple, distinct XenoCorps*. Those organizations can co-exist under different ownerships and offer different pricing schemes or kinds of services, to clients as well as to XenoServers. For instance, a XenoCorp might prefer to pay XenoServers' owners according to the absolute resource usage they report, while another might agree to give a proportion of its revenue to the XenoServer's owner. Also, XenoCorps might decide to use digital cash, direct debit payment after the job has finished, or tickets issued before the job is submitted, in order to perform charging. Even further, tickets can be valid only for a particular XenoServer, or for any of the XenoServers cooperating with the XenoCorp that issued the ticket.

Another major decision that should be taken by each XenoCorp is about the exact *model of interaction* between XenoCorp and the XenoServers that cooperate with it. For example, if tickets that are not tied to a specific XenoServer are used in order to perform charging and authorization of clients, there is a tradeoff between the frequency of updates sent to XenoCorp by the XenoServers and the imposed network traffic. When a job starts being executed on a XenoServer, it could be immediately reported to XenoCorp, which would eliminate the risk of having the same ticket used more than once on several XenoServers. However, this would significantly increase network load and system complexity. On the other hand, the approach of having XenoServers inform XenoCorp periodically about the resource usage on them would limit traffic, but introduce a risk

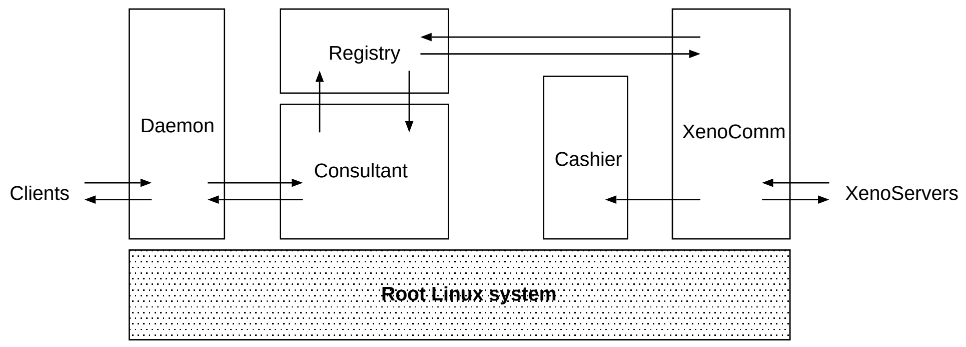

Figure 1: XenoCorp structure

of vulnerability against malicious clients that, for instance, use the same ticket on more than one XenoServers at the same time. We have done some initial work on analyzing the effect that different values for the period of updates would have in the case of fraud [7].

3 Resource Discovery and Management

Executing jobs on any computing system requires physical resources like CPU time and memory. In order to run a job, the operating system needs to locate available resources and then allocate a part of them to the new task to be executed. The term *resource discovery* is used to refer to the problem of finding an appropriate execution context for a job, and *resource management* to denote the process of administering and dividing the available resources between jobs in an efficient way.

In conventional systems, running on a single machine, resource management is being carried out by a centralized *scheduler*, which is a part of the operating system that allocates physical resources to processes according to some criteria. Discovering available resources in monolithic systems is also not much of a problem, as resources are local and the operating system is aware of the exact amount and location of the available resources at any time.

However, in distributed systems there is no centralized point of control. Resources are reserved and released dynamically, network links fail independently and unpredictably, machines and servers connect and disconnect in an arbitrary way. Moreover, locating, administering and sharing distributed resources efficiently and fairly in a global federated system with competing users, like the XenoServers platform, appears to be a major challenge. For all the above reasons, the issues of resource discovery and management in the XenoServers distributed execution platform need to be investigated further.

3.1 XenoCorp

Resource discovery, in the context of XenoCorp, is defined as the process of locating and suggesting the most appropriate XenoServers for a particular job submitted by a client. In other words, XenoCorp is responsible for matchmaking between clients and XenoServers. In the proposed architecture, this task is carried out by the *Consultant* module of XenoCorp servers. The main structure of a XenoCorp server is shown in Figure 1.

There are many parameters that the Consultant has to take into account, while attempting to figure out which XenoServer would be the best choice for a job that has been received by the *Daemon* module. First of all, the candidate XenoServer must fulfill the *environmental requirements* of the client, like a specific version of an operating system or any execution platform. Also, *network traffic parameters* and *server load* should be taken into consideration, as XenoCorp

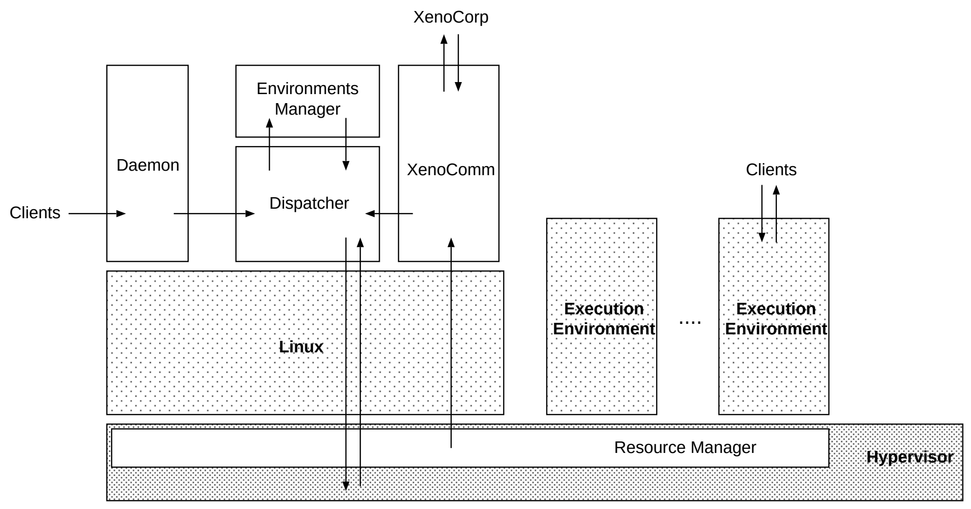

Figure 2: XenoServer structure

should try to perform load balancing. Additionally, XenoCorp should support the functionality of attempting to minimize the cost for the job to be submitted, if the client requests it, by checking and comparing the *pricing schemes* of all XenoServers that are appropriate to host it. At the same time, the *physical location* of XenoServers has to be investigated, as it is a strong intention of our platform to favour geographical proximity of servers and clients. All this information, regarding available XenoServers and their properties, is received by the *XenoComm* module and cached in the *Registry* module in each XenoCorp server, in order to reduce network load and eliminate unnecessary broadcast messages.

We propose that clients should be given the ability to specify the *priorities* that the XenoCorp should obey while searching for a suitable XenoServer. In other words, clients should have the option to specify themselves which properties of a XenoServer they would consider "useful". Then, the Consultant module performs the search and examines all parameters on their behalf. After the search is over, XenoCorp will suggest a number of appropriate XenoServers, found according those criteria, to the client, and supply information about their location, load and pricing schemes. In the end, it is up to the client to choose a XenoServer, and submit its job directly there for execution.

3.2 XenoServers

Jobs that are to be hosted by a XenoServer, have to be launched in the appropriate context. The XenoServer has to execute them on an instance of the operating system they require, and give them some amount of memory, supply them with IP addresses, hard disk space and any other kind of resources they have requested. The set of those resources, which are allocated to an instance of an operating system in order to host clients' jobs, are called an *execution environment*.

In the architecture we propose, illustrated in Figure 2, XenoServers store information about all environments that are currently running or have been running in the past, containing details about the operating system used, the name of the kernel, the root directory, network parameters, memory and CPU that each environment can use and so on. This information is held in the *Environments Manager* (EnvManager) module. When a new job arrives and gets accepted by the *Daemon* [7], the *Dispatcher* module is called, as this is the one responsible for matching the requirements of the job with any of the environments in the EnvManager module. If this procedure succeeds, the job will be dispatched in the environment chosen for execution. If not, a new environment that meets the specifications of the job has to be initialized. This process of matching requirements with environmental specifications can be carried out efficiently using XML pattern matching techniques [6]. This constitutes the *resource discovery* process inside a XenoServer.

XenoServers are able to accommodate large numbers of jobs and execution environments, running at the same time. It is very important that clients get the amount of resources they pay for, and that jobs are not allowed to get any arbitrary amount of resources they ask for. *Resource management* in the XenoServers context is the process of dividing a single set of physical computing resources between several operating system instances, according to the Quality of Service (QoS) guarantees they have paid for.

The *Resource Manager* module of XenoServers is responsible for coordinating and allocating resources to execution environments in a fair way. There are several parameters that have to be taken into consideration by the Resource Manager while distributing resources to execution environments, such as QoS requirements –like maximum completion time, minimum CPU percentage given, minimum QoS crosstalk–, resource congestion or past accounting information. The set of the parameters that influence resource management, and the effect that they will have on the distribution of resources, is defined as the *resource management policy* that a XenoServer follows.

There are studies [12] regarding *dynamic* or *automatic pricing*, proposing the use of pricing as a mechanism to regulate resource congestion. As a resource becomes more congested, its price rises, encouraging users to move to less congested resources, and possibly providing a source of revenue for increasing the capacity of the congested resource. Also, they propose the use of QoS-adaptive applications, which are programs that can adjust the amount and type of resources they require, according to the feedback that they get from the resource manager of the underlying operating system. In our architecture, owners of XenoServers must be given the opportunity to modify and specify their XenoServer's pricing scheme manually, in any way they want. However, some of them might find it more convenient to use dynamic pricing, which will administer resource pricing and congestion automatically and efficiently.

4 Conclusions

The XenoServers infrastructure offers an innovative approach to distributed computing, supporting efficient resource management, accurate accounting and charging, and reliable resource isolation for its clients. The proposed platform changes the traditional interaction models radically, as servers can be moved close to clients on-demand, in a dynamic and efficient way, in order to enforce locality of services, reduce communication latency and improve the utilization of servers. In this paper, we have outlined some design issues of the XenoServers platform, and presented our ideas regarding how resource discovery and management can be performed. A large subset of the described mechanisms are still in the design or early development stage.

5 Acknowledgements

We would like to acknowledge the financial support of Marconi plc for Evangelos Kotsovinos' work, and the members of the XenoServers project for our productive and inspiring cooperation.

References

[1] Anurag Acharya and Joel Saltz. A study of internet round-trip delay. Technical Report CS-TR-3736, UMIACS and Department of Computer Science, University of Maryland, 1996.

[2] Gerco Ballintijn and Maarten van Steen. Characterizing internet performance to support wide-area application development. *ACM SIGOPS Operating Systems Review*, 34(4):41–47, 2000.

[3] E.Gialama, E. Markatos, J. Sevasslidou, D. Serpanos, E. Kotsovinos, and X. Asimakopoulou. DIVISOR: Distributed video server for streaming. In *Proceedings of the 5th IEEE/WSES International Conference on Circuits, Systems, Communications and Computers (CSCC)*, pages 4531–4536, June 2001.

[4] K.A. Fraser, S.M. Hand, T.L. Harris, I.M. Leslie, and I.A. Pratt. The Xenoserver computing infrastructure, a project overview, February 2001.

[5] Keir Fraser. Xenoservers: Service platforms in the internet infrastructure. In *6th CaberNet Radicals Workshop*, February 2002. Funchal, Madeira Island.

[6] Haruo Hosoya and Benjamin Pierce. Regular expression pattern matching for XML. In *Proceedings of the 28th ACM SIGPLAN-SIGACT symposium on Principles of programming languages*, pages 67–80. ACM Press, 2001.

[7] Evangelos Kotsovinos. First year report, July 2002. Computer Laboratory, University of Cambridge.

[8] Klaus Krauter, Rajkumar Buyya, and Muthucumaru Maheswaran. A taxonomy and survey of grid resource management systems for distributed computing. *Software Practice and Experience*, 32(2):135–164, February 2002.

[9] T. S. Eugene Ng and Hui Zhang. Predicting internet network distance with coordiantes-based approaches. In *Proceedings of the 21st Annual Joint Conference of the IEEE Computer and Communications Societies (INFOCOMM 2002)*, June 2002.

[10] Venkata N. Padmanabhan and Lakshminarayanan Subramanian. An investigation of geographic mapping techniques for internet hosts. In *Proceedings of the 2001 conference on applications, technologies, architectures, and protocols for computer communications*, pages 173–185. ACM Press, 2001.

[11] Dickon Reed, Ian Pratt, Paul Menage, Stephen Early, and Neil Stratford. Xenoservers: Accountable execution of untrusted programs. In *Workshop on Hot Topics in Operating Systems*, pages 136–141, 1999.

[12] Neil Stratford and Richard Mortier. An economic approach to adaptive resource management. In *Workshop on Hot Topics in Operating Systems*, pages 142–147, 1999.