Distribution + Persistence = Global Virtual Memory
A Position Paper

Stephen Russell Alan Skea Kevin Elphinstone Gernot Heiser Keith Burston
Ian Gorton Graham Hellestrand

School of Computer Science and Engineering
University of New South Wales, NSW Australia 2033
email: dist@vast.unsw.edu.au

1 Introduction

The Distributed Systems Group at the University of New South Wales is currently constructing a distributed operating system based on *global virtual memory* (GVM). Unlike previously published systems, our system combines local and remote storage into a single large virtual address space. This provides a uniform method for naming and accessing objects regardless of their location, removes the distinction between persistent and transient data, and simplifies the migration of data and processes.

Our GVM system uses conventional computing nodes connected to specialised network interfaces. A fault-tolerant migration and replication protocol keeps the system operational and consistent in case of network errors or node crashes. Password capabilities are used to control access to the GVM.

2 System Overview

The dominant models for current distributed systems are based on communication between objects [6, 14, 28, 30]. Regardless of whether this communication is done explicitly, or is hidden by an RPC interface, a program treats external objects as being potentially remote. The artificial distinction between local and remote objects often requires different naming and access mechanisms.

A similar distinction is made in naming internal data residing within a program's address space, and external data, such as files, managed by a traditional operating system. Persistent programming systems [11, 25] attempt to unify access. However, few of these systems have addressed the problems of distribution, and have been restricted in scale by a limited address space.

With the advent of 64-bit microprocessors, it is now possible to investigate systems in which all objects reside in a *single* global virtual address space. This address space solves the problems of transparent distribution and supporting persistence by removing unnecessary distinctions between local and remote objects. In such a GVM system, exactly the same mechanism would be used to address a local variable of a procedure, for example, as would be used to access a database which resides on a remote node. Data can be shared by reference without unnecessary copying, and references to any object can be embedded in user data structures. As well, process migration involves little more than moving the process' register set to another node.

Our goal is to provide a minimal set of mechanisms to efficiently support distributed object oriented systems. We believe that an operating system should concentrate on issues such as mobility, replication and protection. Other policy decisions about object support should be left to the developers of languages and applications. Our proposed object support mechanisms are described in the following sections. The final section then compares our system with other approaches.

3 Objects and Migration

Our GVM system supports course grained objects consisting of multiple pages. These objects are analogous to files and directories in traditional operating systems. Support for fine grained objects is expected to be provided by language runtime systems. Objects are created by allocating a portion of the global virtual address space. Each node is responsible for managing the portions of the GVM from which it creates objects.

Protection is provided at the object level, and access is authenticated when the object is first referenced. Object protection information is stored at the creation node, and replicated at other nodes.

From the system's point of view, however, the GVM is divided into fixed-size pages which may migrate around the network. A multi-reader/single-writer protocol is used to maintain coherency, with ownership of pages migrating to the writing node. Pages may also be migrated to achieve load balancing.

Each node maintains hints as to the likely current location of the individual pages. Remote pages not found by querying these nodes are located by broadcasting their addresses to the network. These broadcasts could impose an unacceptable load on the node processors, so we are investigating special hardware that supports the object location protocols. A preliminary proposal for this hardware has been made in [29].

4 Protection

In our GVM system, we propose a mixture of hardware and software mechanisms to provide multiple levels of security. At the first level, hardware protection of the network will be provided by a variant of Amoeba's F-boxes [30]. All connections to the network are via physically secure interfaces which use one-way functions to prevent intruders from receiving messages addressed to other nodes.

Protecting network access is not sufficient, however, to provide protection of the objects in the GVM. At the system level, memory access is controlled using password capabilities similar to those used by [1]. A capability consists of the object's global virtual address (GVA) and a password. Every object has a set of different passwords, each of which confers a particular access permission, such as read–only, read/write or executable.

A process can only access those objects that have been mapped into its view of the GVM. Such a mapping may be established by a direct request to the system. Alternatively, the first access to an unmapped page results in a page fault that creates the mapping, provided that the process possesses a suitable capability.

Each group of processes maintains its own list of capabilities, which is searched by the page fault handler. The capability lists are themselves persistent, and so are available immediately when a user logs in. Each list therefore represents the environment of a user.

The system also provides a distributed authentication service which manages protected directories for storing capabilities. While these services have been provided by some existing distributed operating systems [20], many questions arise as to how the existing approaches can be adapted to the GVM model.

5 Persistence and Replication

Our goal is that the proposed GVM system exhibit such a degree of fault tolerance that most users' work will not be seriously affected if a small number of computing nodes crash. It is essential that this level of fault tolerance has little or no impact on system performance.

This requires that all network communication uses a protocol that maintains GVM consistency in the presence of faults. In particular, the page migration algorithm must ensure that page ownership is never lost or shared. It is anticipated that existing approaches [3, 4, 17] can be adapted for our system.

Secondly, we would like to improve performance by making use of distributed backup copies. We intend investigating several different approaches for maintaining the consistency of replicated page copies, including weak consistency models based on object types [2], and using replicants to support a resilient persistent store.

6 Comparison With Other Systems

Some aspects of the above issues have been previously addressed. Our GVM system combines aspects of existing distributed shared memory (DSM) systems, distributed programming systems and distributed operating systems.

Existing DSM systems fall in two categories: distributed multiprocessor systems [5, 16, 17], and systems that provide a shared memory service, maintained either by the kernel [12] or by user-level servers [2, 19].

Systems in the first category are designed for running dedicated multiprocessing applications and provide no multiuser or general purpose computing support. Systems of the second type provide limited shared memory which can only be used by explicit actions on the part of the programmer. By contrast, our goal is to provide a general purpose operating system built on top of the GVM system.

Both classes of DSM systems have other deficiencies. None of the systems address the problems that arise in the case of a node failure, they do not protect the address space from eavesdropping on the network, and there is no fine-grained access control to the shared memory areas. These issues have all been addressed by our system.

There have been many distributed programming systems [4, 6, 18] and distributed operating systems [10, 26, 27, 28] constructed in recent years. While the goal of these systems is similar to our own, they differ in many important aspects. None of these systems present a uniform address space, but instead provide an object based model with remote operations. Each object resides in its own private address space. Remote data can only be accessed indirectly by invoking an operation provided by the owner of the data which returns a copy of the requested data.

Early systems such as Multics [21] and more recent systems such as Domain [15] and Locus [23] allow files (or portions of files) to be mapped into process address spaces. In our GVM system all objects reside in the same global address space, so no explicit mapping is required. Our system also takes advantage of advancing technology to overcome hardware address space limitations that constrained these earlier systems.

Our choice to use password capabilities results in three benefits. First, we can use conventional microprocessors as our node processors, unlike other proposals to support large address spaces [9, 13]. Second, the alternative approach to prevent forgeries of capabilities by segregating them into a separate space [24, 31] prevents users from storing GVAs in their own data structures. Finally passwords effectively increase the sparseness of the address space and allow addresses to be reused.

7 Current Status

The page migration and replication system is currently being designed, and a prototype will shortly be developed using the x-kernel [22]. We have commenced modifications of the Choices [8] operating system. A group of students are instrumenting a Unix system to obtain data for our simulations of page size, address space management and page migration. We are also building specialised network hardware which is a locally developed 110Mbps broadband ring which supports efficient multicast communication [7]. A prototype system should be operational early in 1993.

References

[1] M. Anderson, R. Pose, and C. Wallace. A password-capability system. *Comp. J.*, 29(1):1–8, 1986.

[2] J. K. Bennett, J. B. Carter, and W. Zwaenepoel. Munin: Distributed shared memory based on type-specific memory coherence. In *Conf. Princip. and Pract. Parallel Programming*, pages 168–176. ACM, 1990.

[3] P. A. Bernstein, V. Hadzilacos, and N. Goodman. *Concurrency Control and Recovery in Database Systems*. Addison-Wesley, 1987.

[4] K. Birman. Replication and fault-tolerance in the ISIS system. *ACM Symp. OS Princip.*, pages 79–86, 1985.

[5] R. Bisiani and M. Ravishankar. PLUS: A distributed shared-memory system. In *17th Int. Symp. Comp. Arch.*, pages 115–124. IEEE, 1990.

[6] A. Black. Distribution and abstract types in Emerald. *IEEE Trans. Softw. Engin.*, SE-13(1):65–76, January 1987.

[7] A. Burston. An architecture for a broadband LAN. In *Int. Conf. Private Switching Syst. and Netw.*, London, June 1992. IEE.

[8] R. H. Campbell, V. Russo, and G. Johnston. Choices: The design of a multiprocessor operating system. In *Proc. USENIX C++ W.*, pages 109–123, Santa Fe, USA, 1987.

[9] W. Cockshott and P. Foulk. Implementing 128 bit persistent addresses on 80x86 processors. In J. Rosenberg and J. Keedy, editors, *Int. W. Comp. Arch. to Support Security and Persistence Inform.*, pages 123–136, Bremen, Germany, 1990. Springer-Verlag.

[10] P. Dasgupta, R. LeBlanc, and W. Appelbe. The Clouds distributed operating system. In *Int. Conf. Distr. Comput. Syst.*, 1988.

[11] A. Dearleand, G. Shaw, and S. Zdonik, editors. *Int. W. Persistent Obj. Syst.*, Martha's Vineyard, USA, 1990. Morgan-Koffman.

[12] B. D. Fleisch and G. J. Popek. Mirage: A coherent distributed shared memory design. In *ACM Symp. OS Princip.*, pages 211–223, 1989.

[13] D. Koch and J. Rosenberg. A secure RISC-based architecture supporting data persistence. In J. Rosenberg and J. Keedy, editors, *Int. W. Comp. Arch. to Support Security and Persistence Inform.*, pages 188–201, Bremen, Germany, 1990. Springer-Verlag.

[14] R. Lea, P. Amaral, and C. Jacquemot. COOL-2: an object oriented support platform built above the Chorus microkernel. In *Int. W. Obj. Orient. Operating Syst.*, pages 68–72, Palo Alto, USA, 1991. IEEE.

[15] P. Leach, P. Levine, J. Hamilton, and B. Stumpf. The filesystem of an integrated local network. In *ACM Comp. Science Conf.*, New Orleans, 1985.

[16] D. Lenoski, J. Laudon, K. Gharachorloo, A. Gupta, and J. Hennessy. The directory-based cache coherence protocol for the DASH multiprocessor. In *Int. Symp. Comp. Arch.*, pages 148–59. IEEE, 1990.

[17] K. Li and R. Schaefer. A hypercube shared virtual memory system. In *Int. Conf. Parallel Processing*, pages 125–32. IEEE, 1989.

[18] B. Liskov. Distributed programming in Argus. *Comm. ACM*, 31(3):300–312, March 1988.

[19] R. G. Minnich and D. J. Farber. The Mether system: Distributed shared memory for SunOS 4.0. In *Summer USENIX Conf.*, pages 51–60, 1989.

[20] J. Morris, M. Satyanarayanan, M. Conner, J. Howard, D. Rosenthal, and F. Smith. Andrew: a distributed personal computer environment. *Comm. ACM*, 29(3), March 1986.

[21] E. I. Organick. *The Multics System: an Examination of its Structure*. MIT Press, 1972.

[22] L. L. Peterson, N. C. Hutchinson, and S. W. O. H. Rao. The *x*-kernel: A platform for accessing internet resources. *Computer*, 23(5):23–33, 1990.

[23] G. Popek, B. Walker, J. Chow, D. Edwards, C. Kline, G. Rudisin, and G. Thiel. LOCUS: a network transparent, high reliability distributed system. In *ACM Symp. OS Princip.*, pages 169–177, 1981.

[24] J. Rosenberg and J. Keedy. Object management and addressing in the MONADS architecture. In *Int. W. Persistent Obj. Syst.*, Appin, Scotland, 1987.

[25] J. Rosenberg and D. Koch, editors. *Int. W. Persistent Obj. Syst.*, Newcastle, Australia, 1989. Springer-Verlag.

[26] M. Rozier, V. Abrossimov, F. Armand, I. Boule, M. Glen, M. Guillemont, F. Herrman, C. Kaiser, S. Langlois, P. Leonard, and W. Neuhauser. Overview of the CHORUS distributed operating system. Technical Report CS/TR-90-25, Chorus systèmes, April 1990.

[27] R. Schantz, R. Thomas, and G. Eono. The architecture of the Cronus distributed operating system. In *Int. Conf. Distr. Comput. Syst.*, pages 250–259, 1986.

[28] M. Shapiro, Y. Gourhant, S. Habert, L. Mosseri, M. Ruffin, and C. Valot. SOS: an object-oriented operating system - assessment and perspectives. *Comput. Syst.*, 2(4):287–338, December 1989.

[29] A. Skea. A memory and network interface for a processor in a distributed system. Technical report, School Comp. Science and Engin., University NSW, Sydney, Australia, 1992.

[30] A. Tanenbaum, R. van Renesse, H. van Staveren, G. Sharp, S. Mullender, J. Jansen, and G. van Rossum. Experiences with the Amoeba distributed operating system. *Comm. ACM*, 33:46–63, 1990.

[31] M. Wilkes. Hardware support for memory protection: Capability implementation. *Symp. Architectural Support for Progr. Lang. and Operating Syst.*, pages 108–116, 1982.