Amnon Barak Shai Guday Richard G. Wheeler

The MOSIX Distributed Operating System

Load Balancing for UNIX

Springer-Verlag
Berlin Heidelberg New York
London Paris Tokyo
Hong Kong Barcelona
Budapest

Series Editors

Gerhard Goos
Universität Karlsruhe
Postfach 69 80
Vincenz-Priessnitz-Straße 1
W-7500 Karlsruhe, FRG

Juris Hartmanis
Cornell University
Department of Computer Science
4130 Upson Hall
Ithaca, NY 14853, USA

Authors

Amnon Barak
Shai Guday
Richard G. Wheeler
Institute of Computer Science, The Hebrew University of Jerusalem
91904 Jerusalem, Israel

CR Subject Classification (1991): D.4, C.2.4

ISBN 3-540-56663-5 Springer-Verlag Berlin Heidelberg New York
ISBN 0-387-56663-5 Springer-Verlag New York Berlin Heidelberg

Typesetting: Camera ready by authors/editors
45/3140-543210 - Printed on acid-free paper

Dedicated to our families.

Preface

This book describes the design and internals of the MOSIX distributed operating system. **MOSIX**, an acronym for **M**ulticomputer **O**perating **S**ystem for **UNIX**, integrates a cluster of loosely connected computers into a virtual single-machine UNIX environment. The main property of MOSIX is the *high degree of integration* among the computers, which may include personal workstations, shared memory and non-shared memory multiprocessors, connected by fast communication links. This integration includes network transparency, cooperation between the computers to provide services across machine boundaries, support of dynamic configuration, and system-initiated load balancing by process migration. Another property of MOSIX is the ability to scale up the system configuration to encompass a large number of computers. This is accomplished by using probabilistic algorithms that allow each computer to maintain only partial knowledge about the state of the global system, regardless of the number of computers.

The development of MOSIX was begun in 1981 for a cluster of PDP-11 computers. It was based on UNIX Version 7. Since then, four additional versions of MOSIX have been developed, each version based on the most recent version of UNIX that was available at the time. The latest version is operational on a cluster of workstations, where each workstation is itself a multiprocessor.

We describe MOSIX as it is, rather than as it was meant to be, or as it would be implemented today. The text summarizes some relevant parts of UNIX that provide a basis for understanding MOSIX. Readers interested in detailed descriptions of UNIX should refer to any of the excellent texts referred to in this book. The material presented is intended primarily for readers who are interested in distributed and multiprocessor systems. The reader is assumed to have some knowledge in programming and operating systems, preferably UNIX. Readers without this background will still benefit from the techniques and algorithms discussed.

The book consists of eleven chapters. Chapter 1 gives a brief introduction to MOSIX, placing it in the context of other multicomputer systems. Chapter 2 presents an overview of MOSIX, its characteristics, kernel architecture, and developmental stages. Chapter 3 describes the design of the traditional, non-distributed UNIX file system, followed by Chapter 4, which describes several distributed UNIX file systems and the MOSIX file system in particular. Chapter 5 describes the non-distributed UNIX process structure and the kernel

mechanisms that interact with the processes. Chapter 6 presents the MOSIX process, the internal mechanisms that allow process migration in MOSIX, and the details of the distributed interprocess communication mechanisms. The underlying links between the host-specific part of the kernel and the user view of the kernel are presented in Chapter 7.

Chapter 8 describes the load balancing mechanism built into the MOSIX kernel. Scaling considerations, including examples of probabilistic algorithms that are used in MOSIX, are presented in Chapter 9. Chapter 10 presents the performance of the main communication mechanisms of the MOSIX kernel. Chapter 11 discusses distributed applications. It presents a brief description of a language extension for writing distributed programs and gives the performance results of several examples. It concludes with a description of a monitoring facility for distributed applications that is supported by extensions to the MOSIX kernel.

The development of MOSIX has been a cooperative effort of many individuals, but some call for particular mention here. First, we must acknowledge the significant contributions of Amnon Shiloh to the design and implementation of all the MOSIX versions, and for reviewing drafts of the manuscript. Ami Litman made valuable contributions to the design and development of the first version of MOSIX. Special thanks to Danny Braniss for his help and to Avi Barel for the implementation of NSMOS. Thanks are also due to Robert Hofner, Roy Laor, Jonathan Masel, On G. Paradise, Gil Shwed, Yuval Yarom and all of the other participants on the MOSIX team for their contributions. We would like to thank National Semiconductor Corporation (Israel) for their support and equipment contributions, and Jennifer Steiner for editing the manuscript.

The research that led to the development of MOSIX was supported in part by the U.S. Air Force, Office of Scientific Research, sponsored by the HQ Rome Air Development Center and the European Office of Aerospace Research and Development, the Israel Ministry of Science and Technology, the Israel National Council for Higher Education, the Israel Ministry of Defense, National Semiconductor Corporation, and the Israel Academy of Science and Humanities.

A. Barak, S. Guday, R.G. Wheeler
Jerusalem, 1993

Contents

Chapter 1

Introduction

Configurations of loosely-coupled multicomputers can be classified as either network systems or distributed systems. In a **network system**, each computer runs its own operating system. Each of these operating systems is augmented by communication facilities that permit interaction with the other systems in the network. In a network system, the user's environment is confined to one (local) computer, with the added ability to access objects that reside on other (remote) computers. This last ability, however, is quite limited, since all network commands must specify the location of each remote object. For example, users access remote objects (e.g., reading a file that resides on another computer) by explicitly using the network name of the computer that has the file. As a result, network operating systems have a limited degree of resource sharing and they are therefore commonly used to connect geographically dispersed and heterogeneous systems.

Distributed systems provide a higher degree of transparency and resource sharing than network systems. Distributed systems can be divided into two categories: user-level distributed systems and distributed operating systems. In a **user-level distributed system**, the support for distribution is provided in a layer of software on top of the (non-distributed) operating system. User-level distributed systems are intended for configurations of machines running different operating systems with the distributed software layer on top. An example of this type of distributed system is the Open Software Foundation's Distributed Computing Environment [17]. **Distributed operating systems**, on the other hand, implement support for distribution in the kernel, and are intended for configurations of machines running the same operating system. The MOSIX system described in this book is an example of a distributed operating system.

In a distributed operating system, one operating system is used by all the computers in the entire network, each computer running its own copy. Distributed operating systems are most commonly used in networks in which all of the computers are from the same manufacturer. Distributed operating systems can also be used in networks of computers from different manufacturers if all of the operating systems have the same functionality (i.e., the same user interface).

The main goal of distributed operating systems is to provide resource sharing over a transparent network. In these systems, the user is provided with a single virtual machine, with transparent network communication, distribution of the workload, and automatic resource allocation.

The architecture of many distributed operating systems is based on the **client/server model**. Among these distributed operating systems, it is possible to distinguish between asymmetrical client/server systems and symmetrical client/server systems. In an **asymmetrical client/server system**, specific machines are assigned service functions (e.g., file servers or name servers), while other machines are used to execute user tasks. Examples of asymmetrical client/server distributed operating systems include the V-System [14] and Amoeba [29].

The main characteristic of a **symmetrical client/server system** is the decentralization of control; each machine is both a server and a client for all of the services. In symmetrical systems, each machine can function as an independent computer, with complete hardware and software facilities, while the entire network behaves like a single computer. The advantages of symmetrical systems over asymmetrical systems include improved cost/performance ratio, better resource utilization, increased availability and reliability, and the possibility to scale up the configuration to large numbers of computers.

The MOSIX system is a symmetrical distributed operating system that integrates a cluster of loosely connected, independent computers into a virtual single-machine UNIX environment. The hardware configuration for MOSIX consists of a cluster of computers, each with its own local memory, that are loosely connected by a local area communication network (LAN). In most configurations that have been developed to run MOSIX, each computer (node) is an independent uniprocessor UNIX system, with complete hardware and software facilities. In the latest configuration running MOSIX, each node is itself a multiprocessor. Each such node may contain up to eight independent processors that share I/O devices and communication controllers over a common bus. In this book, the terms "node" machine, and "workstation" are used to refer to an independent computer. The term "processor" refers to a single Processing Element (PE) within a multiprocessor workstation.

The main characteristics of MOSIX are:

Network transparency – the network is completely invisible to the naive user.

Autonomy each node is capable of operating as an independent system.

Cooperation the nodes work together to provide services across the network.

Decentralized control – each processor makes all of its control decisions independently.

Dynamic process migration – processes can be migrated among homogeneous processors.

Load balancing – process migration allows near-optimal assignment of processes to processors.

Dynamic configuration – nodes may be added and removed with minimal side effects.

Increased availability – files and processes can be replicated on different nodes.

Performance – local and remote operations are highly efficient.

Reliability – a limited degree of reliability is provided through the isolation of faults.

Replicated kernel and resources – the system is replicated in each node.

Scalability – the configuration may be scaled up to a large number of nodes.

Compatibility – MOSIX is compatible with AT&T UNIX System V.

More details about these characteristics are given in Chapter 2.

The most noticeable properties for executing distributed applications on MOSIX are its network transparency, the symmetry and flexibility of its configuration, and its dynamic process migration. The combined effect of these properties is that application programs are completely independent of the current state of the system configuration. Users do not need to change their applications due to node or communication links failures, nor be concerned about the load of the various processors. The system automatically attempts to optimize all resource allocation, including migrating I/O-bound jobs to the sites that master the devices they use and migrating heavily communicating jobs to nearby processors so that they benefit from fast services.

The MOSIX kernel is obtained by restructuring the UNIX kernel into machine-dependent and machine-independent parts (modules). The kernel is built as a structure of loosely-coupled modules, where the module interfaces are minimal and well defined [6]. The machine-dependent module provides site-dependent services, such as access to local disks. The machine-independent module provides network-wide services to the application level, such as transparent interprocessor communication. Each MOSIX kernel isolates the users' processes from the specific machines on which they execute, while at the same time providing these processes with the standard UNIX interface [2, 3]. This means that processes execute in a site-independent mode, which allows all system calls to be executed uniformly, regardless of the current location of the requesting process and the site that has the requested object.

The MOSIX kernel is designed to hide the internal network from the user and the application programmer [6]. In order to provide efficient network-wide services, MOSIX kernels interact with each other at the level of kernel remote procedure calls. The MOSIX kernel can be implemented on any reasonable hardware, but the participating processors must be homogeneous, to allow

process-migration. This does not exclude MOSIX from being part of wider, heterogeneous networks, where process migration occurs among disjoined sets of homogeneous processors, improving resource sharing and performance.

Another major objective of the MOSIX kernel architecture is high performance. An important consideration in the design of this kernel was the avoidance of the high overhead caused by information hiding. Another architectural consideration was to enable comprehensive debugging of the system on a single machine. This means that, if necessary, any of the known debugging techniques can be applied to the MOSIX kernel. The details of the MOSIX kernel architecture are given in Chapter 2.

MOSIX belongs to a class of modular operating systems, designed as a set of system servers on top of a **microkernel** that provides low-level services (e.g., memory management and interprocess communication). In this type of operating system, the microkernel forms a standard base that can support higher level system-specific interfaces. The specific interface that is supported by MOSIX is UNIX. Examples of other microkernel-based operating systems include Amoeba [29], BirliX [20], Chorus [21], and Mach [46].

There are currently only a few distributed operating systems that support process migration. This is due to the difficulties of managing the migration itself, the need to change the operating system kernel architecture to support migration, the need to change the environment of the process (e.g., its dependence on local kernel tables), and the need to provide access to these resources from other sites. Examples of other operating systems that support process migration include Rhodos [49] and Sprite [15, 30]. For further reading about the design of distributed operating systems see [18]. A comprehensive survey of distributed systems is given in [12].

The motivation behind the MOSIX project is to research and develop operating systems for message passing-based (as opposed to shared memory) distributed systems. This research began in 1981, with the development of the original version of MOSIX for a cluster of PDP-11 computers. That version was compatible with UNIX Version 7 [47]. The second, M68000-based system was compatible with UNIX Version 7 with enhancements from BSD 4.1. Three additional versions have been developed for National Semiconductor's VR32, the VAX family, and National Semiconductor's VME532 multiprocessor architecture. These versions are compatible with UNIX System V Release 2 [2]. The development strategy for all these versions was to use as many ready-made hardware and software components as possible. This strategy enabled the development team to concentrate on such issues as the kernel architecture, probabilistic algorithms, performance, and debugging aids without spending too much effort on hardware design or the supporting software outside the operating system kernel.