The NOW MOSIX and its Preemptive Process Migration Scheme *

Amnon Barak, Oren Laden, Yuval Yarom
Institute of Computer Science§
The Hebrew University
Jerusalem 91904, Israel

Abstract

Recent advancements in microprocessors and network technologies have made Network of Workstations (NOW) a feasible alternative to parallel computers for high performance, time-sharing and interactive computing. MOSIX is an enhancement of UNIX that allows clusters of computers and workstations to share their resources. It was originally developed as a "single-system" image operating system for clusters of LAN and bus connected, distributed memory multicomputers, where the whole cluster appears to be one system. Recently, MOSIX was re-designed for network of workstation configurations using the "home" model, in which all the user's processes are created and seem to run at the user's (home) workstation. Like all the previous versions, the NOW MOSIX supports efficient, transparent work distribution and load-balancing by preemptive process migration. This creates an efficient, resource sharing execution environment for multiple processes, with maximal performance and high degree of overall resource utilization. This paper describes the latest version of MOSIX for NOW configurations and its preemptive process migration scheme.

Key words: Distributed systems, dynamic load-balancing, high performance systems, network of workstations, preemptive process migration.

*Supported in part by a grant from the MOD.
§ E-mail: {amnon, orenl, yval}@cs.huji.ac.il.
WWW: http://www.cs.huji.ac.il/papers/distrib/

1 Introduction

The increased interest in Network of Workstations (NOW) as an alternative to parallel computers for high performance and interactive computing [1], requires the use of policies for dynamic work distribution. This is necessary to support a flexible use of personal workstations, i.e., to utilize idle workstations and to relieve a workstation when its owner wishes so. The main policies for dynamic work distribution are load balancing and load sharing, that use some form of remote execution and process migration. These policies can perform initial distribution of the work load among the workstations and re-distribution of the work load when the system becomes unbalanced. One mechanism that can transparently perform these tasks efficiently is a preemptive process migration, which combined with load balancing or load sharing can maximize the performance, respond to resource availability and achieve high degree of overall utilization.

In spite of the advantages of preemptive process migration only few operating systems have implemented automated load-balancing [2, 10] or load sharing [8] policies using preemptive process migration. Instead, most existing systems provide explicit user-controlled remote execution and migration, while some other systems provide automated remote executions, but perform preemptive process migration at the explicit request of a user. For a survey of systems that provide process migration see [13]. We note that most parallel programming environments, e.g. PVM [9], P4 [5], Linda [6] do not support preemptive process migration, although some work has been reported for PVM [7].

This paper describes the latest version of MOSIX, a UNIX based multicomputer operating system for a NOW that supports preemptive process migration for load balancing. MOSIX was originally developed as a "single-system" image operating system for clusters of distributed memory, LAN and bus connected computers, where the whole cluster appears to the user's processes as one system. In order to support preemptive process migration, the UNIX kernel was partitioned to site-dependent and site independent parts to allow the user process to execute in a site independent mode. This development required a major restructuring of the UNIX kernel, upon which MOSIX is based, the development of several new features, e.g. a communication protocol between the two parts of the kernel and the use of a unique, network-wide naming as well as the development of a distributed file system that did not exist in UNIX at that time. While the above system image model of MOSIX has an elegant design, supports size scalability and is easy to use for running parallel applications, its implementation became increasingly more difficult. This is due to the frequent updates of UNIX and the unique kernel architecture of MOSIX, which required major modifications of the UNIX kernel. In spite of the complexity of this project, five different version of MOSIX were developed, for different architectures and for different versions of UNIX [2].

Recently, MOSIX was redesigned for a NOW configuration using the "home" model which is similar to the "home node" of Sprite [8]. In this model all the user's processes are created and seem to run at the user's "home" workstation. Unlike the previous versions of MOSIX, where a process could access any resource directly from any location by using its unique (global) name, in the "home" model the resources are addressed relatively to the user's working environment, as defined in each user's workstation. In order to be able to support load balancing it was necessary to enhance the UNIX kernel with mechanisms for preemptive process migration, load-balancing and communication, to allow the process to call the current workstation kernel directly for local services and to communicate with the home workstation for services that require the process environment. These enhancements are implemented at the operating system kernel, without changing the UNIX interface, and they are completely transparent to the application level.

In comparison to the previous versions of MOSIX, the implementation of the NOW MOSIX was relatively simple because it was confined to the development of the process migration and load-balancing mechanisms. Otherwise, MOSIX uses existing mechanisms of UNIX, e.g. NFS, without modifying the internal structure of the UNIX kernel. Execution of processes under UNIX with the MOSIX enhancements creates an efficient execution environment that allows many users to share the resources of the NOW, with maximal performance and efficient overall utilization of the available resources. Experience of executing parallel processes under PVM [9] with the MOSIX process migration enhancements [3], shows a dramatic improvement in the NOW utilization and a significant speedup of parallel applications.

This paper is organized as follows: the next section presents the NOW MOSIX and its unique properties. Section 3 presents the process migration scheme and gives some details about its implementation. Section 4 shows the efficiency of the load balancing algorithms by comparing the performance of MOSIX to the optimal policy for the execution of parallel tasks. Our conclusions and future plans are given in Section 5.

2 What is MOSIX

MOSIX is an enhancement of UNIX that allows distributed-memory multicomputers, including LAN connected network of workstations, to share their resources by supporting preemptive process migration and dynamic load balancing among homogeneous subsets of nodes. These mechanisms respond to variations in the load of the workstations by migrating processes from one workstation to another, preemptively, at any stage of the life cycle of a process. The granularity of the work distribution in MOSIX is the UNIX process. Users can benefit from the MOSIX execution environment by initiating multiple processes, e.g. for parallel execution. Alternatively, MOSIX supports an efficient multi-user execution environment.

The NOW MOSIX is designed to run on configurations that include personal workstations, file servers and CPU servers that are connected by LANs, a shared bus, or fast interconnection networks. In these configurations each workstation (node) is an independent computer, with one or more (shared-memory) processors, local memory, communication and I/O devices. A low-end configuration may include several workstations that are connected by Ethernet. A larger configuration may include additional file and CPU servers that are connected by a high speed ATM LANs. A high-end configuration may include a large number of nodes that are interconnected by a high performance, scalable, switch interconnect that provide low latency and high bandwidth communication, e.g. Myrinet [4].

MOSIX can be implemented for any version of UNIX and for any hardware, but the processors must be homogeneous to allow process-migration. Users interact with the multicomputer via their own workstations. As long as the load of the user's workstation is light, all the user's processes are confined to the user's workstation. When this load increases above a certain threshold level, e.g. the load created by one CPU bound process, the process migration mechanism (transparently) migrates some processes to other workstations and to the CPU servers.

2.1 The main properties of MOSIX

MOSIX supports the UNIX interface and all its mechanisms. It also supports:

- **Network transparency** - for all cross machine operations, i.e. for network related operations, the interactive user and the application level programs are provided with a virtual machine that looks like a single machine. When a process issues a system call, it is the responsibility of the local kernel (in the machine where the process resides) to perform network-wide operations to execute the call.
- **Dynamic load balancing** - that initiates process migrations in response to load fluctuations and resources availability, to improve the performance of multiple processes by distributing the load evenly among the workstations. The main algorithms of the load-balancing policy are the load calculation algorithm, that measures the local load; the information dissemination algorithm and the (competitive) migration consideration algorithm, that makes the final decision based on the available load information, the relative speed of the nodes and other parameters [2]. We note that all the above algorithms are executed by each node, in a distributed manner and they are not synchronized.
- **Preemptive process migration** - that can migrate any user's process, transparently, at any time, to any available node. The migration itself involves the creation of a new process structure in the remote site, followed by a copy of the process page table and the "dirty" pages. After a migration there are no residual dependencies other than at the home workstation. The process resumes its execution in the new site by few page faults, which bring the necessary parts of the program to that site.
- **Decentralized control and symmetry** - that exploits the redundancy of the hardware to achieve a high degree of availability, without using a master-slave relationships between the workstations. The organization of the system data bases is completely decentralized and no machine possesses information on all the system objects of any type. Control is also decentralized. Each workstation is capable of operating as an independent system and makes all its control decisions independently. This property allows a dynamic configuration, where workstations may join or leave the network with minimal disruptions.
- **File system** - MOSIX uses standard NFS.

The most noticeable properties of executing applications on MOSIX are its network transparency, the symmetry and flexibility of its configuration, and its preemptive process migration. The combined effect of these properties is that application programs do not have to know the current state of the system configuration. This is most useful for time-sharing and parallel processing systems. Users need not recompile their applications due to node or communication failures, nor be concerned about the load of the various processors. Parallel applications can be executed on MOSIX by simply creating many processes, just like a single-machine environment. The operating system automatically attempts to optimize the process allocations and balance the load.

2.2 The system image model

The system image model of MOSIX is a NOW, in which all the user's processes seem to run at the user's "home" workstation. Each new process is created at the same site(s) of its parent process, i.e. a fork of a migrated process creates copies of the *deputy* and the *body*. All the processes of each user have the execution environment of the user's home workstation. Processes that migrate to other (remote) workstations use local resources if possible, but interact with the user's environment through the user's workstation.

2.3 Compatibility

The NOW MOSIX is compatible with BSDI's BSD/OS Release 2.0 [11], which is based on Release 4.4BSD-Lite from the Computer Systems Research Group at UC Berkeley. The current implementation is for Intel's X86/Pentium based workstations and PC's.

3 Process Migration

This section presents the process migration scheme of the NOW MOSIX. The central theme of this scheme is the observation that a UNIX process can be divided to two contexts, the user context - that can be migrated, and the system context that is "home-site" dependent and can not be migrated. We first describe these two parts of the UNIX process, then present the *deputy*, the part of a process that can not be migrated.

3.1 The UNIX process

In MOSIX as in UNIX, a process is the basic computational unit for the execution of a program [12]. A UNIX process has two contexts, the user context and the system context. The user context contains the program code, stack and variables of the process. The system context contains description of the resources which the process is attached to, and a stack for the execution of system code on behalf of the process.

The process interacts with its environment using two mechanisms, system calls and signals. System calls enable processes to operate on resources. In order to inform the process of the occurrence of exceptional or asynchronous events, the system employs signals. Whenever such an event occurs the appropriate signal is sent to the process.

3.2 The Deputy

The main requirement for a process migration is transparency, that is, the functional aspects of the system's behavior should not be altered as a result of migrating a process. Achieving this transparency requires that the system is able to locate the process, e.g. in order to preserve the user's interface; and that the process is unaware of the fact that it has been moved from one node to another. A simple way to achieve these two requirements is to maintain in the "home" site of the process a structure that represents the process and interacts with the process environment. In MOSIX, the "home" site of the process is the user's workstation. The structure that represents the process in that site is called the *deputy*.

The concept of the *deputy* of a process is based on the observation that only the system context of a process is site dependent. Generally the idea is that the user context of a process is migrateable, and since its interface to the system context is well defined, it is possible to intercept every interaction between the user and system contexts, and forward this interaction across the network.

In order to be able to migrate a process, it was divided to two parts: the *body* which has all of the user context and almost none of the system context of the process, and the *deputy* which has the rest of the system context, and no user context. The user context and the part of the system context that is associated with the *body* are site independent. The *body* can, therefore, be migrated to any node. The *deputy* has the site dependent part of the system context of the process, hence it must remain in the home site of the process. The two parts are connected by a communication channel, on which interaction between the two parts takes place. Figure 1 shows two processes that share a home site. One process is unsplit (process A) and the other is a split process (process B) that has been migrated.

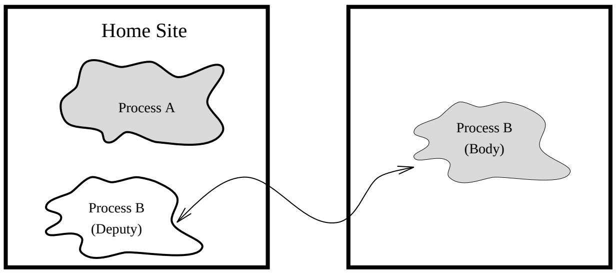

Figure 1: *Unsplit, and split processes*

In the execution of a process in MOSIX, location transparency is achieved by forwarding site dependent system call to the home site. System calls that are executed by the *body* are intercepted by the remote site's kernel. If the system call is site independent it is executed locally, otherwise, the system call is forwarded to the *deputy*. The *deputy* executes the system call on behalf of the *body* in the home site, and sends the result back to the *body*, which then continues to execute the user's code.

The home site *deputy* approach offers a simple solution to the process location requirement. Instead of providing the location of the process, the *deputy* provides a channel to interact with the *body*. The kernel at the home site informs the *deputy* of asynchronous events, the *deputy* checks if there is any action to be taken, and informs the *body* if so. The *body* checks the communication channel for reports of asynchronous events in the same locations a standard UNIX process checks for signals. The *deputy* approach is also a very appealing method to achieve a clean solution to the transparency requirement. This solution is robust, and is not affected even by major modifications of the UNIX kernel. Some modifications, such as supporting a distributed file system, might even improve the solution by making many system calls site independent. Also, this solution seems to be portable, it relies on almost no machine dependent features of the kernel, and thus does not hinder its porting to different architectures. In fact, it does not rely much on UNIX specific properties, and seems to be exportable to other time-sharing operating systems.

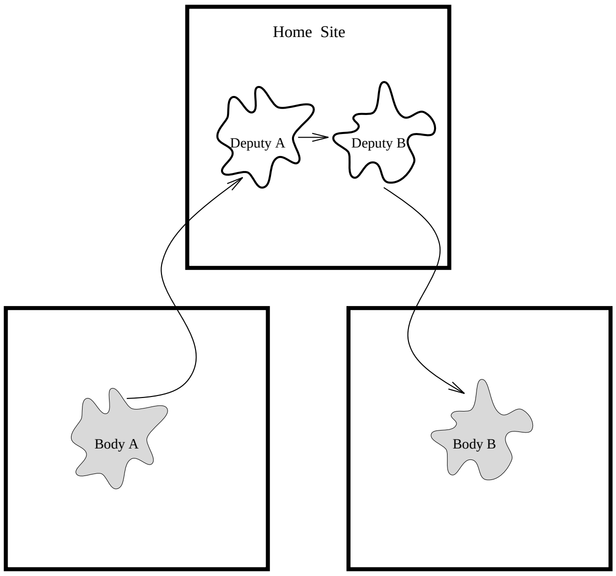

Figure 2: *The path of data sent from process A to process B*

The *deputy* approach has, however, some overhead in the execution of system calls. For example, as Figure 2 shows, data sent from one process to another has two hops over the network, instead of just one hop. This implies some scaling limitations due to a bottleneck at the home site, when executing a large number of communicating processes. Additional overhead is incurred due to the need to hold enough state at the home site, and to keep a dedicated communication channel.

Nevertheless, we believe that all of the above problems could be solved by the development of additional mechanisms. These mechanisms should make some of the resources globally available, thus reducing the number of site dependent (remote) system calls and improve the overall performance. Given the current memory prices, the memory requirements overhead due to the need to keep the communication links and the deputies is negligible. Idle deputies do not require much CPU resources, and by reducing the number of site dependent system calls, the deputies will be idle most of the time. Also, the bottleneck of handling many communication channels at the home site can be solved by the development of migrateable sockets, that can be migrated with one of the processes that uses the socket.

4 Performance of the Load Balancing Scheme

This section presents the efficiency of the MOSIX process migration algorithms. We executed a set of identical CPU-bound processes and compared the execution times to the theoretical execution times of the same processes using the optimal preemptive process migration algorithm. The execution platform is a NOW configuration with 16 identical Pentium workstations that are connected by an Ethernet. We note that in the load-balancing scheme of MOSIX, a process is migrated only when the difference between the loads of two nodes is above the load created by one CPU bound process. This policy differs from the time-slicing policy commonly used by shared-memory multicomputers.

Table 1 presents the results of this benchmark. The execution time of each process was set to 300 seconds. The first column lists the number of processes. The second column lists the theoretical execution times (in seconds) of these processes, assuming no communication overhead and a preemptive process migration when the difference between the loads of two nodes is greater than the load created by one CPU bound process. Column three lists the measured execution times (in seconds) of the processes using the MOSIX load balancing algorithm, and column four the slowdown ratio (in %) between column three and column two.

No. of Procs.	Optimal Time	MOSIX Time	Slow-down(%)
1	300	301.91	0.6
2	300	302.92	1.0
4	300	304.57	1.5
8	300	305.73	1.9
16	300	310.83	3.6
17	450	456.91	1.5
20	450	462.07	2.7
24	450	471.87	4.9
25	525	533.15	1.6
27	525	549.07	4.6
31	563	574.03	1.9
32	600	603.17	0.5
33	700	705.93	0.8
36	700	715.35	2.2
38	750	759.90	1.3
40	750	767.67	2.4
43	833	833.33	0.0
47	883	901.81	2.1
48	900	916.11	1.9

Table 1: Optimal vs. MOSIX load balancing performance

From Table 1 it follows that the average slowdown ratio of the MOSIX load balancing algorithm vs. the optimal execution algorithm is 1.95%. We note that the MOSIX load balancing scheme imposes a minimal residency (local) execution time for each new process before it can be migrated, to prevent short-lived processes from migration. This residency time is the main reason for the difference between the results of column two and column three in Table 1.

5 Conclusions and Future Plans

The NOW MOSIX has been operational for over 2 years on a cluster of 22 Pentium and several i486 based workstations. It is used for research and development of multicomputer systems and parallel applications. Its unique mechanisms provide a convenient environment for writing and executing parallel programs, with minimal burden to the application programmers.

Currently we are researching the idea of migrateable sockets to overcome potential bottlenecks of executing a large number of communicating processes. We are also developing competitive, on-line algorithms to improve the performance of the load-balancing policy. After we install the Myrinet LAN [4], we intend to start several new projects that benefit from the pool of resources of the NOW. One project is to develop a *memory server* that can swap portions of a large program to idle memory in remote workstations. This mechanisms could benefit from our process migration mechanism, that is capable to page across the network. This project is similar to the network RAM project described in [1]. Another project is to develop a shared memory mechanism and to use it for communication among groups of processes.

Beyond that, we intend to make the NOW MOSIX available on the Internet, as an enhancement of BSD/OS for parallel computing. For information contact the first author.

References

[1] T.E. Anderson, D.E. Culler, and D.A. Patterson. A Case for NOW (Networks of Workstations). *IEEE Micro*, 15(1):54–64, February 1995.

[2] A. Barak, S. Guday, and R.G. Wheeler. The MOSIX Distributed Operating System, Load Balancing for UNIX. In *Lecture Notes in Computer Science, Vol. 672*. Springer-Verlag, 1993.

[3] A. Barak and O. Laden. Performance of PVM with the MOSIX Preemptive Process Migration. Technical Report 95-08, Computer Science Institute, The Hebrew University, March 1995.

[4] N.J. Boden, D. Cohen, R.E. Felderman, A.K. Kulawik, C.L. Seitz, J.N.Seizovic, and W-K. Su. Myrinet: A Gigabit-per-Second Local Area Network. *IEEE Micro*, 15(1):29–36, February 1995.

[5] R. Butler and E. Lusk. User's Guide to the p4 Programming System. Technical Report TM-ANL-92/17, Argonne National Laboratory, October 1992.

[6] N. Carriero, E. Freeman, and D. Gelernter. Adaptive Parallelism and Piranha. *Computer*, 28(1):40–49, January 1995.

[7] J. Casas, D. Clark, R. Konuru, S. Otto, R. Prouty, and J. Walpole. MPVM: A Migration Transparent Version of PVM. Technical Report CSE-95-002, Oregon Graduate Institute of Science & Technology, February 1995.

[8] F. Douglis and J. Ousterhout. Transparent Process Miration: Design Alternatives and the Sprite Implementation. *Software – Practice and Experience*, 21(8):757–785, August 1991.

[9] A. Geist, A. Beguelin, J. Dongarra, W. Jiang, R. Manchek, and V. Sunderam. *PVM - Parallel Virtual Machine: A User's Guide and Tutorial for Networked Parallel Computing*. MIT Press, Cambridge, MA, 1994.

[10] G.W. Gerrity, A. Goscinski, J. Indulska, W. Toomey, and W. Zhu. Rhodos-A Testbed for Studying Design Issues in Distributed Operating Systems. In *Toward Network Globalization (SICON 91): 2nd International Conference on Networks*, pages 268–274, September 1991.

[11] R. Kolstad, T. Sanders, J. Polk, and M. Karles. *BSD/OS 2.0 and BSDI Internet Server Release Notes*. Berkeley Software Design, Inc., Colorado Springs, CO, 1995.

[12] J. Leffler, M.K. McKusick, M.J. Karels, and J.S. Quarterman. *The Design and Implementation of the 4.3BSD UNIX Operating System*. Addison-Wesley, Reading, MA, 1989.

[13] M. Nuttall. Survey of Systems Providing Process or Object Migration. Technical Report DoC 94/10, Imperial College, London, May 1994.