OPERATING SYSTEM FOR PARALLEL COMPUTING

A.Y. Burtsev, L.B. Ryzhyk
Institute for Applied System Analysis

When available computers are incapable of providing a sufficient computing power for solving the arising tasks, while purchasing new and more powerful equipment is economically infeasible, then the only way to gain additional computing power is to parallel the computational process between several interconnected computers. The idea of parallel distributed computing has been studied in cybernetics since early sixties. However, until recently parallelism was primarily employed within the framework of a single computer only. For example, it formed the basis for extending the classical von Neumann architecture to superscalar processors and later – to explicit parallel instruction computing (EPIC) architectures [1].

The majority of computational tasks can be implemented in concurrent manner. The highest level of parallelism is achieved when the task can be represented by a set of loosely coupled sequential processes that have only few synchronization points.

Modern operating systems support parallel execution of processes on multiprocessor and uniprocessor computers (the latter form of parallelism is known as pseudo-parallelism). For this purpose an operating system provides process synchronization and communication facilities. However, a process is limited to a single computational node. In order to implement a parallel algorithm in a multicomputer environment application programmer has to involve additional libraries and tools for organizing synchronization and data exchange between remote processes. Obviously, this task can be more effectively addressed by an operating system.

Recently, we started the work on a new operating system aiming to support transparent execution of parallel algorithms in a distributed environment. This system will provide the distribution mechanisms, which will allow a developer to migrate his parallel application from a single computer to a network with minimal efforts. The new system will support dynamic balancing of a computational payload among network nodes. For this purpose it will provide each process with execution environment which will not be bound to a specific machine. Thus, a process started in one node can be transparently transferred to another node at any time. This procedure known as *process migration* will form the basis for building the distributed execution environment.

Currently there exist a number of approaches to carrying out computations in distributed environments. Most of them rely on special parallel programming libraries that provide application developers with message-based distributed communication and synchronization facilities. Two well-known examples of such libraries are Message Passing Interface (MPI) [2] and Parallel Virtual Machine (PVM) [3]. This approach allows easily building a distributed programming environment on the basis of a network of computers running conventional operating systems like UNIX or Windows NT. However, parallel programming libraries do not support process migration between network nodes. Besides, these libraries are based on high-level operating system services, which makes it easy to implement them but negatively affects their performance. The most important drawback inherent to MPI, PVM and other programming platforms of this kind is that each of them imposes its own specific process interaction mechanisms. As a consequence, adaptation of a parallel program for execution in a distributed environment requires significant efforts for porting it to a new set of communication primitives.

An interesting alternative approach was taken by MOSIX [4] project aimed to build an extension of the Linux kernel, which would support distributed computing by presenting processes with a single execution environment within a system of several interconnected computers. MOSIX implements efficient process placement and migration policies and provides a distributed shared memory facility.

In contrast, the Sprite [5] project developed a distributed operating system from scratch. Sprite operating system supports process migration and load balancing, i.e., dynamic placement of processes at the least loaded workstations. In Sprite a migrated process executes the majority of system calls locally, while only several location-dependant system calls are transparently forwarded to the process's home node. Sprite also provides a highly optimized *exec* system call, which implements a very fast migration of a process to remote node on its creation. Other Sprite features include distributed shared memory, distributed UNIX IPC mechanisms and high-performance distributed file system.

The main goal of our research is the development of a distributed operating system providing processes with location-independent execution and communication environment and supporting the migration of processes between network nodes. The process migration facility will allow to significantly improve utilization of the computational resources of the network. In most cases migration will occur on process creation: the parent process remains in the home node and the newly created process is transferred to a remote machine. This kind of migration is the fastest one, since the new process does not yet have any allocated virtual memory regions associated with it. However, late migration should also be supported. Upon migration a process can continue using external devices and files located in the home node.

Consider below how an execution context of a process is transferred during migration.

Virtual memory. Virtual memory of the process is an essential part of its state that has to be transferred. The time consumed by virtual memory copying to a target node has a crucial impact on the overall efficiency of migration. The straightforward approach implemented in systems like LOCUS [6] and Charlotte [7] lies in transferring the whole virtual address space of the process to the target node at the time of migration. However, this approach requires freezing the process for a significant period of time. In order to minimize migration latencies, systems like Accent [8], V [9] and Sprite [5] use various lazy address space transfer strategies.

I/O. A single network-wide I/O space, providing processes with uniform access to all peripherals of the distributed system, will be used. Global uniqueness of port names can be guaranteed by attaching a prefix containing the network address of the host to each name. It should be noted that migration of a process that heavily relies on the hardware resources of its home node is often inexpedient.

Files. A distributed file system providing global access to data stored on file servers will be used. The development of such file system is a separate nontrivial task. Therefore the new operating system will most probably rely on one of the existing distributed file systems.

Message channels. Message exchange is a standard interprocess communication facility provided by all modern operating systems. Normally, each process has incoming and outgoing message queues associated with it. These queues must be moved together with migrating process to the target node. Besides, a distributed message exchange system requires global process naming and location mechanism.

System calls. It is desirable to achieve such level of location independence that after migration a process can execute all system calls locally at its new host. Therefore, it is necessary to minimize the number of residual dependencies remaining after migration. However, experience of other systems shows that it is impossible to get rid of all residual dependencies, hence certain system calls will be forwarded to the process's original host.

Another important question to be answered is who and when can make a decision about migration of a certain process. Whenever possible, migration must occur on creation of a process. However, late migration is also necessary for implementing efficient load balancing. In certain cases migration can be manually initiated by application programmer. More often, however, transfer of a process is scheduled by a special load balancing service that tracks the load of all nodes of the distributed system and gathers statistics of process behavior.

Our project will be based on FreeBSD operating system kernel. FreeBSD is a freeware POSIX-compatible version of UNIX, which provides excellent performance and reliability.

An important component of parallel computing environment is an interprocess communication system, providing facilities for data exchange and synchronization. UNIX IPC system includes three mechanisms: shared memory, semaphores, and messages. These mechanisms will be extended to work in a distributed environment. It should be noted that relatively high network latencies can make distributed IPC a performance bottleneck of entire system. Therefore, a special care must be taken while designing network protocols, implementing distributed shared memory, semaphores and messages.

[1] L. Gwennap. *Intel, HP Make EPIC Disclosure*. Microprocessor Report, 11(14), Oct. 1997. pp.5-9

[2] Message Passing Interface Forum. *MPI: A Message Passing Interface Standard*, Mar 1994

[3] A. Geist, A. Beguelin, J. J. Dongarra, W. Jiang, R. Manchek, and V.S. Sunderam. *PVM 3 user's guide and reference manual.* Technical report ORNL/TM-12187, Oak Ridge National Laboratory, May 1993

[4] Barak, A., Shai, G., and Wheeler, R.G. *The MOSIX Distributed Operating System: Load Balancing for UNIX.* Springer Verlag, Berlin, 1993

[5] F. Douglis and J. Ousterhout. *Process Migration in the Sprite Operating System.* Proceedings of the 7th international conference on Distributed Computing Systems, September 1987

[6] G. Popek and B. Walker, editors. *The Locus Distributed System Architecture*. M.I.T. Press, Cambridge, Massachusetts, 1985

[7] Artsy, Y and Finkel, R. *Designing a process migration facility: The Charlotte experience.* IEEE Computer, 1989, pp.47-56

[8] Rashid, R., and Robertson, G., *Accent: A Communictaion-Oriented Network Operating System Kernel*. Proceedings of 8th SOSP, ACM Press, New York, 1981, pp. 64-85,

[9] D.R. Cheriton and W. Zwaenpol. *The distributed V operating system and its performance for diskless workstations.* In Proceedings of the Ninth ACM Symposium on Operating Systems Principles, pp.128-140, October 1983.