

# PLURIX, A DISTRIBUTED OPERATING SYSTEM EXTENDING THE SINGLE SYSTEM IMAGE CONCEPT

R. Goeckelmann, M. Schoettner, S. Frenz and P. Schulthess  
*Department of Distributed Systems, University of Ulm, 89069 Ulm, Germany*  
*goeckelmann@vs.informatik.uni-ulm.de*

## Abstract

The Plurix project implements a lean and highspeed object-oriented Operating System (OS) for PC clusters. Communication between the cluster nodes is achieved via shared objects in a Distributed Shared Memory (DSM), which is organized as Distributed Heap Storage (DHS) containing both data and code.

The consistency of the DHS is guaranteed by the strong transactional consistency model. The latter uses an optimistic synchronization scheme and restartable transactions, which ease the development of distributed applications. The OS (including kernel and drivers) is written in Java using our proprietary Plurix Java Compiler to translate Java source code directly into Intel machine instructions.

Process migration in traditional OSs is a difficult task as parts of the kernel state need to be saved and restored. The Plurix DHS extends the well-known Single System Image (SSI) concept by sharing almost all data and code. In this paper we present relevant parts of the Plurix architecture and show how process migration is implemented and simplified.

**Keywords:** Software Engineering, High-Performance Computing, Operating System

## 1. INTRODUCTION

Ever since IVY [1] the first Distributed Shared Memory (DSM) prototype DSM systems have mainly been developed to support parallel algorithms. Most of them are running on top of traditional Operating Systems (OS) such as Linux, Microsoft Windows or Mach. Data is exchanged through message passing (e.g. MPI) or remote invocation (e.g. RPC, RMI) mechanism.

Typically each node in a cluster runs its own OS with different configurations and contexts. It is mostly unknown which libraries and resources will be available on the next node – making process and object migration difficult.

To attenuate this issue Single-System-Image (SSI) computing architectures have been the mainstay of high performance computing for many years.

A system implementing the SSI concept offers a global and uniform view on available resources and programs for each user. Each node in the cluster provides the same libraries and services, which is important for load balancing and processes migration.

## 2. THE PLURIX OPERATING SYSTEM

Plurix implements a distributed OS for PC clusters. Numerous page-based DSM systems have been implemented, e.g. IVY, Treadmarks [2]. All of them faces the false sharing syndrome which causes page trashing. Plurix alleviates this problem by an object relocation facility based on a bookkeeping of references [3].

Plurix works in a fully object oriented fashion and is entirely written in Java. Access to device registers is not possible in standard Java but unconditionally needed by the OS. To support operating system and hardware level programming we have developed our own Plurix Java Compiler (PJC) with appropriate language extensions. The compiler directly generates Intel machine instructions and initializes runtime structures and code segments in the heap.

### 2.1. Memory Management

The SSI concept requires that all programs and libraries are available on all nodes in the cluster. Plurix satisfies this requirement by organizing the DSM memory as a heap and storing both data and code in the DHS. This reduces redundancy concerning code segments and simplifies the administration of the system. New programs are directly compiled into the DHS using the PJC. The benefits of a heap organization for a DSM have also been discussed in Murks [4].

Memory within the DHS is accessed like local memory without any special allocation functions. The transfer of DHS-objects between cluster nodes is managed within the page-based DSM and takes advantage of the Memory

Management Unit (MMU) hardware. The MMU detects page faults if an object on a non present page is accessed. For each page fault a separate network packet which contains the address of the missing page (PageRequest) is created and broadcast to all nodes in the cluster (Fast Ethernet LAN). The current owner of the page sends it to the requesting node.

Heap management is an important topic in Plurix. It is carefully designed to alleviate false sharing and to reduce collisions during object allocation, e.g. it is designed to let each node allocate memory in a different part of the DSM[5]. Additionally, vital classes and objects of the system receive special protection.

## 2.2. Consistency Model

Consistency of shared and replicated objects is important in distributed systems. In Plurix this is synonymous to the consistency of the entire DHS, which is achieved using a novel consistency model, denoted *transactional consistency* [7]. To ensure consistency all commands are automatically encapsulated into transactions by the OS. At the start of a transaction, write access to all pages is prohibited. If a page is written, the system creates a shadow image and enables write access. The addresses of all modified pages are logged and at the end of a transaction (commit phase) they are broadcasted. Each partner node in the cluster compares the received WriteSet with the ReadSet of the currently running transaction. If a collision is detected, the transaction is aborted and will restart later. In this case all modified pages are discarded and the previous state of the node is reconstructed, using the saved shadow images. A token mechanism makes sure that only one node at a time enters the commit phase.

## 2.3. Scheduler

The transactions used in Plurix can be functionally compared to traditional processes. We have adopted the cooperative multitasking model from the Oberon system [8]. Each node contains a transaction loop executing a number of registered transactions with different priorities. A task or process switch can be achieved by passing to the next transaction enqueued in this loop. Each transaction should be short to reduce collision probability. The lean design of the OS and its components make short transactions possible. Long running computations should be explicitly split into several transactions by the programmer.

## 2.4. Checkpointing and Recovery

In 1990 Fuchs [9] started to adapt recovery strategies originally designed for message passing systems to DSM systems. Numerous subsequent papers discuss the adaptation of checkpointing strategies[10], but the more sophisticated solutions have not been evaluated in real DSM implementations because checkpointing is difficult to achieve in PC-clusters even under global coordination.

The main reason is that if a checkpoint needs to be saved it is not sufficient to save the DSM content but also the local process context needs to be observed. Otherwise errors may occur during recovery, e.g. unsaved and not restored kernel locks. This is not a trivial task because the architecture of the underlying OS has not been designed for cluster operation and checkpointing. Consequently some projects, e.g. Kerrighed modify the OS kernel to alleviate these problems [11]. Nevertheless, there are still limitations and there is a considerable overhead during checkpointing.

## 2.5. Extended SSI Concept

By extending the SSI concept we avoid these architectural drawbacks [5]. Storing the kernel and its data in the DSM makes it easy to save all required data. Local memory is only used for a few state variables for the network device drivers and -protocol and for the so called Smart-Buffers, which help to bridge the gap between non restartable interrupts and transactions [7]. Furthermore, rollback in case of an error is not very difficult in Plurix because the OS can reboot in 250ms and if a reboot is not necessary the OS is designed to be restartable anyway because of the transaction-based processing.

If the OS kernel runs in the DSM space, parameter passing between applications and kernel is elegant and all objects can be used as parameters. There are no references between different address spaces and since all device drivers reside in the DSM the problem of moving recompiled drivers from the DSM into the kernel space vanishes. Because the code segments of the kernel methods are in the DSM code duplication is avoided [5].

## 3. PROCESS MIGRATION

Process migration (and agent technology) is frequently discussed in the context of distributed systems. There are two main reasons for process migration. The major one is load balancing for CPU intensive computations. Another condition for process migration occurs if an user decides to work at an other node and wants to take along his current application context.

### 3.1. Requirements for process migration

To migrate a process from one node to another it is not sufficient to transfer the program code of the process. Typically, processes access data in the memory and on disk. Therefore the entire context of the process has to be saved, transferred and restored. This includes CPU register values, the current stack, open IO-connections and kernel contexts. As a consequence most distributed systems which implement process migration have to modify the OS kernel, but even then process migration is hard to handle.

Register values can be transferred by using the Task State Segment (TSS) of the IA32 architecture. As the TSS includes the start of the stack it can also be transferred with little effort, as only the TSS has to be adjusted on the target node. Transferring data in the heap is not very difficult but often time consuming, as all data have to be transferred regardless if they are needed or not. The more difficult part is the migration of open IO-connections.

In traditional systems processes contain a number of IO-connections – among other things the code of the process is loaded from a file on disk. IO-connections can be classified in three categories. The first one contains IO-connections to files, which are needed to access data on disk. Secondly there are communication channels to other nodes for synchronisation or data exchange. Lastly there are IO-connections to other hardware devices such as human interface devices. Such IO-connections are typically not needed for time consuming calculation processes.

### 3.2. Process migration in Plurix

Instead of using processes Plurix uses transactions which are anchored in a transaction vector for each node. The persistent DSM contains all data which are needed by the transaction and the code of it. Therefore it is not necessary to explicitly transfer heap data from one node to another in case of a transaction migration. Required data pages are transported by the DSM protocol. As the persistence of the DSM is assured by the PageServer [12], transactions need not read or write data on disk. All needed data is available by the DSM so there are no open IO-connections to files. All communication between the cluster nodes is achieved by the DSM, therefore open IO-connections for inter process communication do not exist. As a consequence most transactions in the Plurix context do not need IO-connections.

The execution of a transaction is realized by invoking its run method. Owing to the Java programming language a transaction can not allocate data structures on the stack. Consequently no data from the transaction is found on stack or in processor registers after the transaction has ended and the stack is essentially empty between two

transactions. Transactions which do not use IO-connections can therefore be easily migrated from one node to another. It is sufficient to remove the transaction from the transaction vector of the source node and insert it into the vector of the destination node.

The transaction vector of each node resides in the DSM and is anchored in the Station-Object. Member variables of this object can be accessed by the keyword STATION followed by the variable name e.g. STATION.transactions as if they were static variables in a virtual class [6]. If STATION is used, the compiler generates a special access scheme to the Station-Object, using the address stored at a specific memory location in the local address space as reference to it. Additionally all Station objects can in principle be accessed by each node using the Station vector anchored in the global accessible class Cluster (see figure 1). Especially time consuming calculations normally do not access devices, hence migration of open IO-connections does not belong to this kind of transactions.

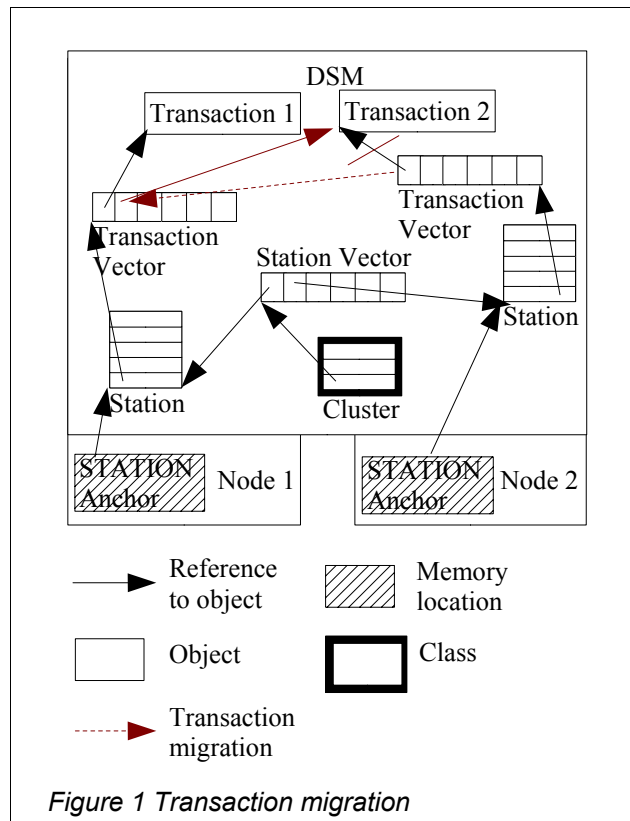


Figure 1 Transaction migration

### 4. Migration of open IO-connections

Transactions communicate with a device through a specific device driver, which is anchored in the STATION-record of this node. A transaction will always access the local instance of this device, which means that

if the transaction migrates to an other node, it will use the device local to this node.

Transactions with open IO-connections to devices are typically user interactive. Such a transaction is only transferred to an other node if the user has logged in on this node and after the environment of this user is restored on the new one. This especially pertains to the window manager. The command transactions can be transferred in the same way as computation intensive transactions.

User interactive transactions need access to at least keyboard, mouse and screen. A few transactions such as controlling transactions for devices like the sound card may access other devices.

Input from the keyboard and output to the screen will be intrinsically local. Therefore the input and output connection is transferred to the new node. As the transactions communicate with the devices through the local STATION instance, this migration is automatically achieved.

Typical interactive transactions are editors for text or images or web browsers. As editors normally access files they typically contain open IO-connection but as in Plurix all data is stored in the DHS, even such transactions does not have IO-connections.

A bit more difficult is the migration of a web browser. Open IO-connections to partners outside the cluster are typically realized via IP. Such a connection can be kept alive by using mobile IP, which is easy to realize by a non migrateable transaction on the home node which copies the data from the network into a buffer residing in the DSM.

For other devices such as a CD-burner a migration of the connection to the new node must not occur. The device driver for such a device will not migrate and only the controlling transaction might migrate. For such devices the transaction should write commands into a command buffer and another, not migrateable transaction reads the command buffer and transfers commands to the device driver. Devices on remote nodes in the cluster can thus be made available by installing a specific transaction on the node hosting the device.

## 5. EXPERIENCES

By moving the kernel into the DHS and thereby refining the SSI concept we achieve a type-safe kernel interface and avoid references between multiple address spaces. All OS methods may be invoked in an object oriented fashion. Taking snapshot of the system is now simple and even process respectively transaction migration of calculating transactions is easily achieved without additional mechanism. Only device controlling transactions for critical devices like a CD-burner need special handling. If such a transaction should be migrateable, it must be split into two parts.

The current version of Plurix runs very stable in the networked cluster environment and without excessive collisions. The current allocation strategy minimizes false-sharing and collisions within the allocation management.

## 6. REFERENCES

[1] K. Li, "IVY: A Shared Virtual Memory System for Parallel Computing", in: Proceedings of the International Conference on Parallel Computing, 1988.

[2] Amza C., Cox A.L., Drwarkadas S. and Keleher P., „TreadMarks: Shared Memory Computing on Networks of Workstations“, in: Proceedings of the Winter 94 Usenix Conference, pp. 115-131, January 1994

[3] R. Goeckelmann, S. Frenz, M. Schoettner, P. Schulthess, "Compiler Support for Reference Tracking in a type-safe DSM", in: Proceedings of the Joint Modular Languages Conference, Klagenfurt, Austria, 2003.

[4] Markus Pizka and Christian Rehn, "Murks - a POSIX threads based DSM system", in: Proceedings of the International Conference on Parallel and Distributed Computing Systems, Anaheim, USA, 2001.

[5] R. Goeckelmann, M. Schoettner, S. Frenz, P. Schulthess, "A Kernel Running in a DSM - Design Aspects of a Distributed Operating System", to appear in: Proc. of the IEEE International Conference on Cluster Computing, Hong Kong, 2003.

[6] R. Goeckelmann, M. Schoettner, M. Wende, T. Bindhammer, and P. Schulthess, "Bootstrapping and Startup of an object-oriented Operating System", in: Proceedings of the European Conference on Object-Oriented Programming - 5th Workshop on Object-Oriented Programming and Operating Systems, Malaga, Spain, 2002.

[7] M. Wende, M. Schoettner, R. Goeckelmann, T. Bindhammer, P. Schulthess, "Optimistic Synchronization and Transactional Consistency", in: Proc. of the Workshop on Distributed Shared Memory on Clusters, in: Proceedings of the IEEE International Symposium on Cluster Computing and the Grid, Berlin, Germany, 2002.

[8] N. Wirt and J. Gutknecht: „Project Oberon“, ACM Press, Addison-Wesley New York 1992.

[9] KunLung Wu and W. Kent Fuchs, „Recoverable Distributed Shared Virtual Memory“, in: IEEE Transactions on Computers, 39(4):460-469, April 1990.

[10] C. Morin and I. Puaut, "A Survey of Recoverable Distributed Shared Virtual Memory Systems", in: IEEE Transactions on Parallel and Distributed Systems, Vol. 8, No. 9, Sep. 1997.

[11] <http://www.kerrighed.org>

[12] S. Frenz, "Persistenz eines transaktionsbasierten verteilten Speichers“, diploma thesis at the University of Ulm, Germany, 2002.

CCECE 2004 – CCGEI 2004, Niagara Falls, May/mai 2004 0-7803-8253-6/04/\$17.00 (c) 2004 IEEE