# Clustix: a Cluster Operating System Based on the Shared Memory Paradigm

Gaël Utard* and Christine Morin†

IRISA/INRIA, Campus de Beaulieu
35042 Rennes Cedex - France

{*gutard,cmorin*}*@irisa.fr*

June 22, 2002

## Abstract

*We propose to study the use of the DSM mechanism to build a distributed kernel for cluster. We think that sharing kernel data with a variable grain in a COMA way could bring some benefits in enabling a single system image operating system on a cluster. With a such design, it could be easier to propose global resource management schemes while offering high availability.*

## 1 Introduction

In 1986, Kai Li published his PhD dissertation entitled "Shared Virtual Memory on Loosely Coupled Multiprocessors" [5]. Since that, benefits and drawbacks of DSM have been largely discussed [1]: its transparency and ease of use is associated with a cost.

We propose to study the relevance of using DSM mechanisms in a new application field: the operating system kernel. In fact, running an operating system on top of a DSM is one of the manners for building a single system image on a cluster.

After showing the globalization issues that a DSM implementation can make simpler, we review some drawbacks of existing single system image designs. Then we give some clues on how to design a distributed kernel based on DSM mechanisms. As a conclusion, we discuss about the interests of such an approach.

*IRISA/INRIA

†IRISA/Université de Rennes I

## 2 Cluster Resource Management

There are two main issues in cluster resource management. The first one is to provide single system image. The second one is to efficiently manage the whole set of resources.

### 2.1 Single System Image (SSI)

An application does not directly see physical resources (processors, memory, disks, network adapters). Through the operating system, it sees a set of logical resources (processes & threads, memory spaces, files, sockets). If each cluster node runs its own operating system, logical resources remain local to each node. It has some implications. First, two processes can not communicate through a shared file or a shared memory area. Second, a process migration should arise, a process would loose its whole environment. Third, users and programmers have to deal with resource distribution.

An elegant way to solve this problem is to present a collection of physical resources as a unique and shared resource. For example, a global file hierarchy can be enabled on a cluster, allowing each node to access to any file (cf fig. 1). Such an illusion is called *single system image* (SSI).

If the illusion of a single system can be enabled at system call level, a lot of SSI [4][6]

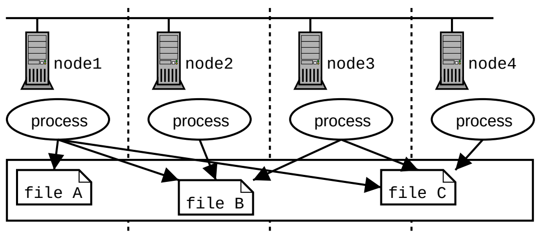

Figure 1: *File hierarchy shared by processes*

enable it in lower layers of the kernel (for example at virtual file system level for file system concerns). With a DSM mechanism, all data is shared, so there is no more *illusion* of a single system, but a real single system.

### 2.2 Global Resource Management

To achieve reasonable performance for a general purpose cluster operating system, global management of all resources is needed. It includes processors, memory, disks and network interfaces. For example (cf fig. 2), the blocks of a file can be distributed on different disks, allowing parallel accesses, and thus an higher banwidth.

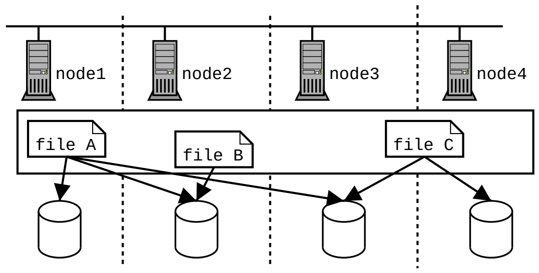

Figure 2: *File blocks stripped across disks*

In interactive mode, with mixed parallel/sequential applications, this management can become somewhat difficult. Let us take an example. There are two applications running concurrently. One needs a lot of processing power and thus is parallel. The other one is sequential but needs a large I/O bandwith. The first application should run on processors of several nodes in the cluster. In the same time, and without slowing down this first application, the second should take advantage of the aggregation of disk bandwidth of several nodes.

Some complications arise at each interface between two resources. If the memory of a node is full, there are several possible actions that can be taken:
- remote paging (injection of pages to memory of remote nodes),
- swap to disk (local or not),
- process migration to a node with free memory.
Before implementing complex policies, the first requirement for a cluster operating system to manage efficiently memory resource, is to handle these three solutions. Mosix [2] is able to migrate processes regarding to their memory usage, thus enabling some kind of cluster memory management. However, in some cases, it is better to do remote paging and Mosix fails to do the best global (CPU and memory) resource management because it does not propose remote paging mechanism.

## 3 SSI Background

We can identify several drawbacks in existing single system image designs.

### 3.1 Global environment

Mosix [2] provides preemptive process migration on top of a cluster, enabling dynamic load balancing. The kernel interface remains unmodified. Thus the use of CPU power and memory of several nodes is completely transparent to the application level.

The most important drawback for Mosix, is that it does not really enable single system image. A file available on node A is not necessary available on node B. However, a process that opened a file on node A and was migrated to B should be able to continue to read it. Thus Mosix has to redirect most of the system calls to the home node (cf fig. 3).

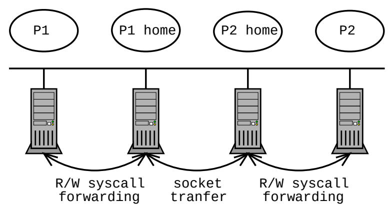

Figure 3: *Syscall redirection*

Moreover this is the case, not only for files, but for communications too. If two processes are communicating and migrate, the exchange of packets will necessary cross their two home nodes.

## 3.2 Data migration

Gobelins [6] proposes a new concept called container. Containers provide the abstraction of shared memory segments at the cluster scale. By integrating this generic sharing mechanism within the host system, between drivers (low-level) and system services (high-level), it is possible to give the illusion to the kernel that system services rely on a physically shared memory.

A container that comes between disk drivers and the read/write interface of the file service enables easy implementation of cooperative file caching. Another container that comes between physical memory management and virtual memory management enables easy implementation of memory sharing.

This concept facilitates process migration. Neither process code, nor process data needs to be explicitely transfered. It is sufficient that the process remains linked to the container, that will handle page transfers on page faults. However, process state is not completely contained in its user data. There is a lot of kernel data structures (e.g. process descriptor, open files descriptors) that have to be transfered (or at least recreated) on destination node. The same problem arises in Solaris-MC [4].

# 4 Cluster Cache

Obviously, running a cluster operating system on top of a page-based shared virtual memory would lead to heavy false sharing and would exhibit catastrophic performance. Thus, the DSM mechanisms have to be carefully adapted to kernel specificities.

## 4.1 Granularity of Sharing

To reach maximal performance, we have to take care of data semantic. Each resource (e.g. a file) is described through specific data structures (e.g. an inode). Kernel data is globally the set of these structures. Each of them is about ten or hundred bytes long and, semantically, it makes up a whole. As example, here are some of the most important structures:

- process descriptor
- open files
- inode
- mount points
- process address space
- memory area
- console descriptor
- socket

The key point is that sharing a logical resource (e.g. a file) corresponds to sharing its descriptors (e.g. corresponding inode). Managing kernel data with this *variable* granularity avoids false sharing.

## 4.2 COMA-like data management

We propose to manage data in a COMA way, forming a big pool of shared kernel structures, cached on multiple nodes. We call this pool of data *cluster-cache* (cf fig 4).

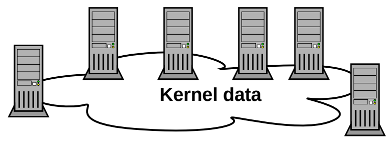

Figure 4: *Cluster-cache*

The cluster-cache requires the use of the sequential consistency protocol. A write invalidate protocol suits, but other protocols could be developed to decrease consistency management cost.

## 4.3 Kernel Modifications

Given the cluster-cache, the design of a distributed kernel for a cluster raises issues similar to the ones faced when designing a traditional SMP kernel. Thus, a distributed kernel can be developed by patching an existing centralized one. However a set of adaptations must be performed:

**Devices** naming has to become global. This requires to adapt minor/major scheme. Moreover read and write primitives for block and char devices have to be slightly modified to enable inter-node data transfers.

**Memory management** has to be adapted to support inter-node page transfer, consistency (in an Shared Virtual Memory[5] way), remote paging and global replacement algorithm (e.g. GMS [3] one).

**Scheduling** remains local, but a set of cooperative kernel threads can initiate process or thread migration. If the migration decision is a complex task, the migration itself consists only in task queue modification. Then all kernel and user data will be automatically transfered through cluster-cache and page cache.

### 4.4 Parallel Services

Given the cluster-cache and these kernel modifications, parallel services become trivial to implement. The parallel file system is a good example. Given the possibility to access remote devices and given the consistency management of page cache, implementing a parallel file system with efficient *cooperative* file caching only requires to implement a software RAID module (that is some device/block number translations through mathematical functions).

## 5 Discussion about interests

We plan to build a prototype of such a "shared memory" distributed kernel. Called Clustix (for CLUSTer and unIX), it should be based on an existing kernel, enabling programmers to use a traditional POSIX API on top of a cluster. In Clustix kernel, all is global and data have a very high mobility. This leads to very interesting features for distributed resource management: remote fork, process migration, remote signaling, remote paging, memory area sharing, parallel file system, etc... Thus Clustix should be an excellent basis to study global resource management algorithms.

The second facet of this high kernel data mobility is that it can be exploited to achieve high availability. First, in case of migration of a sequential process, if great care is taken to preemptively migrate all related data, the process can survive the crash of its home node, unlike most implementations where a deputy remains on this home node. Second, if kernel data is periodically saved on a stable storage support (globally, or only data related to some processes), it can be easily redeployed on safe nodes in the event of a node failure.

## References

[1] Brendan Tangney Alan Judge, Paddy Nixon, Vinny Cahill and Stefan Weber. Overview of distributed shared memory. Technical report, 1998.

[2] A. Barak, O. La'adan, and A. Shiloh. Scalable Cluster Computing with MOSIX for LINUX. In *Linux Expo '99*, 1999.

[3] M. Feeley, W. Morgan, F. Pighin, A. Karlin, H. Levy, and C. Thekkath. Implementing global memory management in a workstation cluster. In *SOSP*, pages 201–212, 1995.

[4] Yousef A. Khalidi, Josep Bernabeu, Vlada Matena, Ken Shirriff, and Moti Thadani. Solaris MC: A multi computer OS. In *USENIX Annual Technical Conference*, pages 191–204, 1996.

[5] Kai Li and Paul Hudak. Memory coherence in shared virtual memory systems. *ACM Transactions on Computer Systems (TOCS)*, 7(4):321–359, 1989.

[6] R. Lottiaux and C. Morin. Containers: A sound basis for a true single sytem image. In *First IEEE/ACM International Symposium on Cluster Computing and the Grid*, 2001.