

# A Practical Comparison of Cluster Operating Systems Implementing Sequential and Transactional Consistency

Stefan Frenz<sup>1</sup>, Renaud Lottiaux<sup>2</sup>, Michael Schoettner<sup>1</sup>,  
Christine Morin<sup>2</sup>, Ralph Goeckelmann<sup>1</sup>, Peter Schulthess<sup>1</sup>

<sup>1</sup> Ulm University, 89069 Ulm, Germany, frenz@vs.informatik.uni-ulm.de

<sup>2</sup> IRISA/INRIA, 35042 Rennes, France, renaud.lottiaux@irisa.fr

**Abstract.** Shared Memory is an interesting communication paradigm for SMP machines and clusters. Weak consistency models have been proposed to improve efficiency of shared memory applications. In a programming environment offering weak consistency it is a necessity to worry about individual load and store operations and about proper synchronization. In contrast to this explicit style of distributed programming shared memory systems implementing strong consistency models are easy to program and consistency is implicit. In this paper we compare two representatives: Kerrighed and Plurix implementing sequential and transactional consistency respectively. Kerrighed is a single system image operating system (OS) based on Linux whereas Plurix is a native OS for PC clusters designed for shared memory operation. The measurements presented in this paper show that strong consistency models implemented at the OS level are competitive.

## 1 Introduction

Many projects in the distributed systems area have aimed at simplifying the development of applications. The proposed systems typically fall into two main categories: message passing and shared memory approaches. Message passing systems typically use explicit data distribution, exchange and synchronization, e.g. MPI, RMI, .NET.

Shared memory libraries implement implicit communication and can automatically guarantee consistency for all objects stored within the distributed shared memory (DSM). For the latter numerous weak memory consistency models have been proposed to minimize synchronization and improve efficiency [3]. Unfortunately, these consistency models put an additional burden on the programmer. Explicit synchronization primitives, like *acquire* and *release*, must be used very carefully and the programmer has to reason about single load and store operations.

In this paper we describe and compare two Operating Systems (OS) implementing a page-based DSM at the kernel level [1], [2], [5], [7]. Kerrighed is a single system image OS based on Linux whereas Plurix is a native OS for PC clusters designed for shared memory operation. Both OSs implement a strong consistency model. Kerrighed implements sequential and Plurix transactional consistency. The measurements discussed in this paper show that strong shared memory consistency models can be efficient and convenient when implemented at the kernel level.

## 2 The Kerrighed DSM

Kerrighed is a single system image (SSI) operating system based on Linux for high performance computing on clusters. For the users and programmers it creates the illusion that a cluster is a single shared memory multiprocessor machine. The Kerrighed DSM is based on a global memory management service implementing the concept of containers.

The key idea is that a container creates the illusion to system services that the cluster physical memory is shared as in an SMP machine. In a cluster, each node executes its own operating system (OS) kernel, which can be coarsely divided into two parts: (1) system services and (2) device managers. We propose a generic service inserted between the system services and the device manager layers called *container* [7]. Containers are integrated in the core kernel thanks to *linkers*, which are software pieces inserted between existing device managers and system services and containers.

Several services, such as the virtual memory service, in a core kernel rely on the handling of physical pages. Linkers divert some functions of these services to ensure data sharing through containers. To each container is associated one or several high level linkers called *interface linkers* and a low level linker called *input/output linker*. The role of interface linkers is to divert device accesses of system services to containers while an I/O linker allows a container to access a device manager.

A container is a software object storing and sharing data between stations. A container is a kernel level mechanism and it is completely transparent to user level software. Data is stored in a container at the request of the host OS of one node and can be shared and accessed by the host OS of other cluster nodes. Pages handled by a container are stored in page frames and can be used by the host kernel as any other page frame. Container pages can for instance be mapped in a process address space.

By integrating this generic sharing mechanism into each host system, it is possible to give the illusion to the kernel that it is managing and using physically shared memory. On top of this virtual physically shared memory the traditional services offered by a standard operating system can be extended to the cluster scale. The existing OS interface is preserved while taking advantage of the low level local resource management mechanisms implemented by the standard node OS.

The containers implement a sequentially consistent memory model using a write invalidation protocol. The memory I/O linker ensures input and output of physical memory pages in and out of containers.

When a container is linked to a memory I/O linker, it becomes a memory container. The memory I/O linker is very simple since it consists in allocating and releasing page frames like the host kernel does for the management of memory segments.

Everything together provides the sight of a single SMP machine, even though the processors are distributed on several nodes in a cluster. The nodes are connected by standard hardware, which is Fast Ethernet for the measurements presented in this paper.

More details on Kerrighed DSM can be found in [8].

### 3 The Plurix DSM

The Plurix project implements a native distributed OS for PC clusters customized for DSM operation. Instead of using special functions for allocating data in DSM memory the Plurix DSM is managed as a heap and accessed like local memory. The benefits of a heap organization have also been identified in other systems but Plurix goes one step beyond by also storing code and runtime structures in the DSM. Thus we extend the SSI concept by storing OS, kernel, and all drivers in the DSM.

Distributed garbage collection relieves programmers from explicit memory management. Unreferenced objects can be collected very easily using the compiler-supported bookkeeping of references [4].

Because weaker consistency models are hard to program and because weak consistency might jeopardize OS integrity we have introduced a strong model called transactional consistency. Memory pages are distributed and read-only copies are replicated in the cluster. When writing to a memory page all read-only copies are invalidated and the writing node becomes the new owner of that page. Inconsistencies are avoided by synchronizing memory accesses from different nodes using our transactional consistency model [5].

In contrast to existing memory consistency models we do not synchronize memory after each write access but bundle several operations within a transaction (TA). In case of a conflict between two transactions we rely on the ability to reset changes made by a TA. This conflict resolution scheme is known in the database world as optimistic concurrency control. Optimistic concurrency control occurs in three steps: the first step is to monitor the memory access pattern of a TA. For this purpose we use the built-in facilities of the memory management unit (MMU) of the processor.

The next step is to preserve the old state of memory pages before modifications. Shadow images are created, saving the original page state before the first write operation within a TA. These shadow pages are used to restore the memory in case of a collision, as described in the next step.

During the validation phase of a terminating TA the access patterns of all concurrent TAs in the cluster are compared. In case of a conflict at least one TA is rolled back using the shadow pages otherwise the latter are discarded.

Currently, we have implemented “first-wins” collision resolution using on a circulating token. Only the current owner of the token is allowed to commit. During a commit the write-set of the TA is broadcast to all nodes in the Fast-Ethernet LAN. All nodes in the cluster compare the write set with their running TA to detect conflicts and to abort voluntarily. In future we plan to integrate other conflict resolution strategies to improve fairness.

Instead of having traditional processes and threads the scheduler in Plurix works with transactions. We have adopted the cooperative multitasking model from the Oberon system, [6]. In each station there is a central loop (the scheduler) executing a number of registered transactions with different priorities. Any TA can register further transactions. System TAs, e.g. the garbage collector, are automatically registered by the OS. Furthermore, the OS automatically encapsulates all user commands within a transaction.

Transactions should be short to minimize collision probability. For long running transactions like the tested calculations, the programmer has to split the calculation in multiple steps appropriate to transactions.

## 4 Comparison of Sequential and Transactional Consistency

In this section, we present a performance evaluation of Kerrighed and Plurix. To evaluate performance of both systems, we used two parallel applications: SOR and ray tracer, programmed using a shared memory paradigm.

### 4.1 Experimental Platform

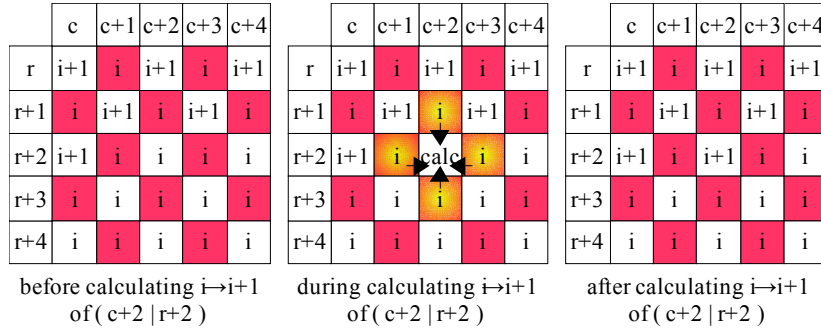
The measurements for both systems have been carried out using 12 nodes:

- Single AthlonXP 2500+ 1833 MHz
- Asus A7V8X-X mainboard with KT400A chipset
- 512 MB DDR-333-RAM
- 3Com 905 B and C Fast Ethernet network cards
- Allied Telesyn AT-8024 switch

The Plurix cluster doesn't require any cluster-outside connection, but Kerrighed needs an additional NFS-server for the shared file system, which is connected in the same manner. In both settings there was no additional traffic on the cluster-network during measurements.

### 4.2 Successive-Over Relaxation (SOR)

A single  $n,n$ -matrix with randomized shared data is transmitted at the start of calculation and then changed during iterated calculation, where border elements are not changed. The matrix is red-white-coloured, and the calculation of the next iteration is done in two phases, where first all white and then all red elements are iterated. The calculation of an element (see figure 1) requires the four bordering neighbours, so all source-values are derived from the same iteration-step:



**Fig. 1.** Calculation of an Element Accessing Its Four Element-Neighbors

Distribution is achieved by splitting up calculation in bands of lines. There is no intrinsic write-conflict on elements, but there is the need of synchronization after each phase of iteration because of the small overlapping read-area of one line at top and bottom frontier, where reading across the borders needs the other values to correspond to the reader's phase. Figure 2 shows the calculation of line  $r$ , which requires read-access of lines  $r-1$  and  $r+1$ . Synchronization is achieved with barriers both in Kerrighed and Plurix.

	0	1	2	3	4	...	n-4	n-3	n-2	n-1
r-2	k	i+1	i	i+1	i	...	i	i+1	i	k
r-1	k	i	i+1	i	i+1	...	i+1	i	i+1	k
r	k	calc	i	calc	i	...	i	calc	i	k
r+1	k	i	i	i	i	...	i	i	i	k
r+2	k	i	i	i	i	...	i	i	i	k

accessed red fields during calculating white fields  
of row  $r$  (constant outline fields  $k$  with  $n,n$ -matrix)

**Fig. 2.** Calculating a Line Accessing Neighbor-Lines

If  $r$  resides at the top border of a node's band, line  $r-1$  is inside the band of the previous node, and if  $r$  resides at the bottom border of a node's band, line  $r+1$  is inside the band of the next node. These nodes will calculate their elements in parallel, so on page-based distribution such as with Kerrighed and Plurix, there is read access to elements that reside on pages, which are written by another station in the same phase. But as all nodes start calculating with their first line of their band, this is not an indispensable bottle neck: calculation takes long enough to disperse accesses to first line of node  $t+1$  and to last line of node  $t$ , so nodes running Kerrighed with a MESI-like protocol for pages do not get in the way of nodes. Within Plurix reads and writes occur atomically during the commit of a TA, so each phase is split up in two sub-phases for bisection of calculated bands. Thereby the borders are not crossed mutual, so the nodes do not read and write concurrently to the same line in the same phase.

Because of synchronization after each phase and because all nodes calculate the same amount of phases, all nodes finish after nearly the same time. The synchronization is not very data intensive and therefore mainly depending on network latency. In contrast the network bandwidth is important for data exchange after synchronization, because for each node two rows of the matrix (for the measurements one row is between 8 and 28 kilobytes) have to be transmitted.

SOR is implemented from scratch both for Plurix (java compiled with the Plurix Java Compiler) and Kerrighed (C compiled with gcc) based upon the Splash-II-Suite.

### 4.3 Ray Tracer

The calculation of the ray tracer starts with an empty shared result-matrix as container for the image. Each node calculates each element independently from other nodes or elements based upon a shared scene definition supporting spheres and triangles with colored and reflective surfaces as well as multiple and different light sources. For this measurements, the scene-definition (see picture RAY) contains 99 spheres and 8 triangles illuminated by three light sources. Each result pixel can be calculated without information about other pixels and as a consequence there is no need of synchronization and the result-matrix is write-only during calculation. Apart from the transfer of the scene definition and of the accesses to the result-matrix there is no intrinsic communication and therefore the ray-tracer demonstrates the limits of system distribution and system dependent scaling. The ray tracer is implemented from scratch both for Plurix (using the Plurix Java Compiler) and Kerrighed (C compiled with gcc) based upon project 5 of class 6.837 at MIT [14].

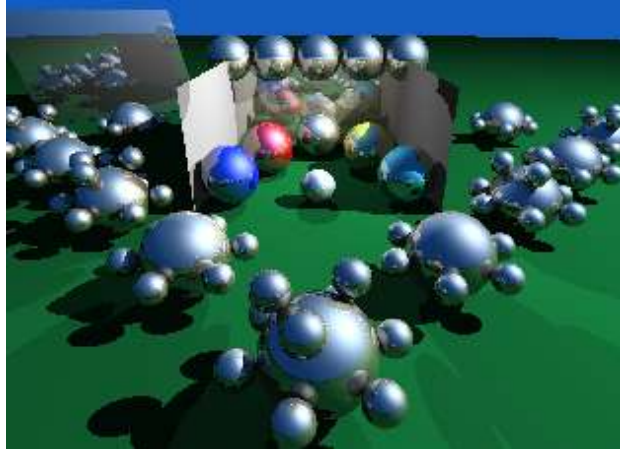


Fig. 3. Calculated Scene

#### 4.4 Experimental results with SOR

The SOR-algorithm has been tested on both systems using 1, 2, 4, 8 and 12 nodes and with the following matrix sizes: 2048x2048, 3584x3584, 4096x4096, 5068x5068, 6144x6144, 7168x7168. Measurement results are presented in tables SSK and SSP. Figure 4 presents the speed-up of the SOR algorithm on Kerrighed. Figure 5 presents the speed-up on Plurix.

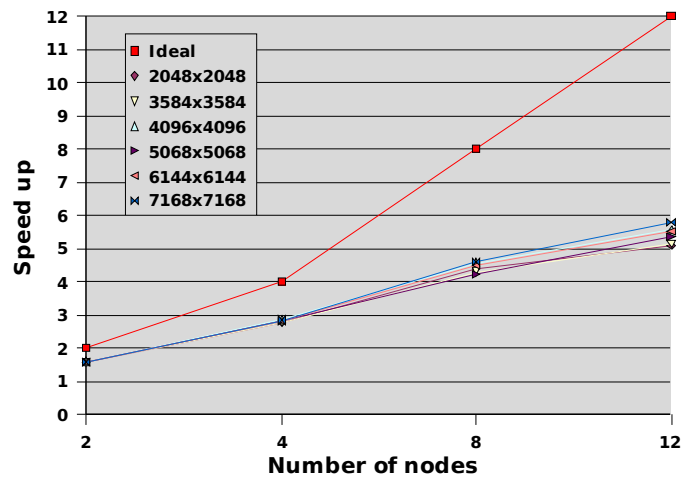


Fig. 4. Speed-up with SOR on Kerrighed

We can observe on fig. 4 and fig. 5 that the matrix size has little impact on speed-up for Kerrighed but more so for Plurix, which will be explained later. The best speed-up achieved is 5.8 (Kerrighed) respectively 6.7 (Plurix) on 12 nodes, which is far from ideal.

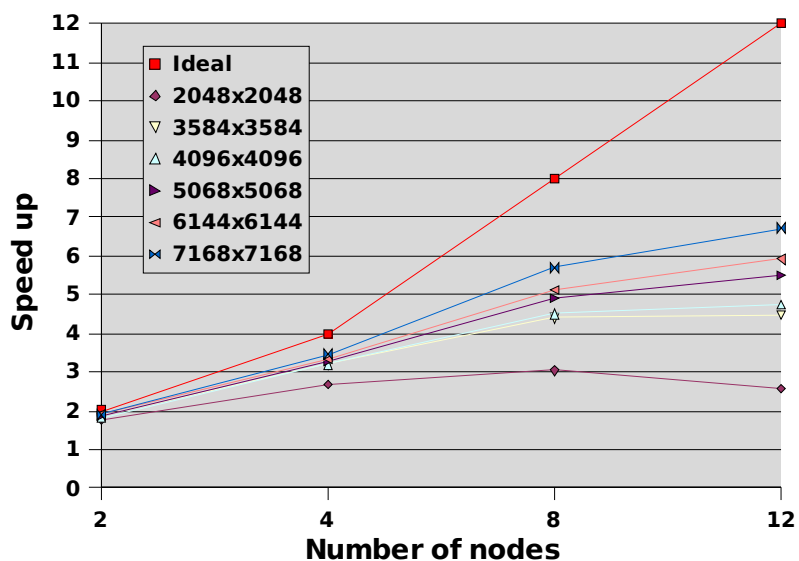


Fig. 5. Speed-up with SOR on Plurix

The main reason for less than ideal performance on both systems is the large gap between processor speed and network bandwidth. While the processors are very powerful, the network is comparatively slow. The SOR algorithm exchanges border rows between each phase of the computation inducing communications. On the available hardware platform, the network-bandwidth/processor-speed ratio is not good enough to reach high speed-ups. Some experiments using a faster network such as gigabit Ethernet or Myrinet would be of interest.

The main difference between the systems is not the difference in speed-up for a large matrix but the different behavior for a small matrix: even with the smallest matrix Kerrighed has a speed-up on 12 nodes similar to the largest matrix, whilst Plurix has a point of reversal on 8 nodes. This is because of the synchronization mechanisms used on Plurix, that are expensive compared to the few calculations that have to be done for small matrices. The barrier implementation on Plurix is in a not fully developed state and still subject of research.

#### 4.5 Experimental results with Ray Tracer

The ray tracer has been tested on both systems using 1, 2, 3, 4, 6, 8, 10 and 12 nodes and with the following image sizes: 2048x1536, 4096x3072 and 5792x4344. Measurement results are presented in tables RSK and RSP. Figure 6 presents the speed-up of the ray tracer on Kerrighed. Figure 7 presents the speed-up on Plurix.

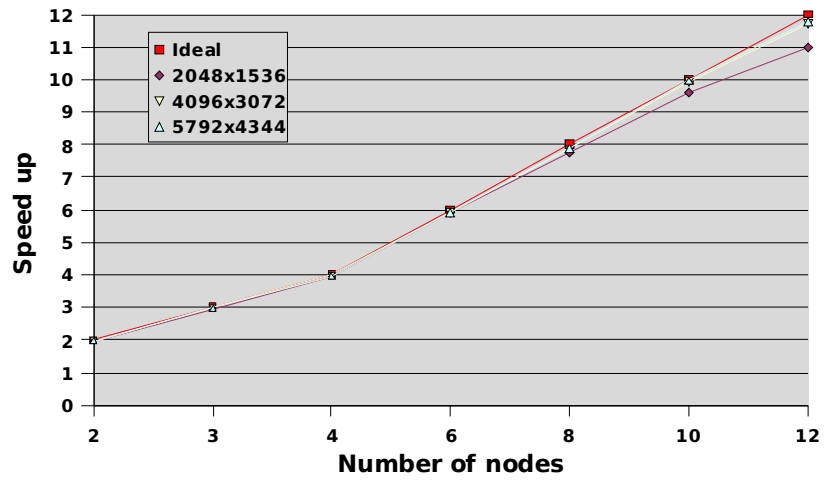


Fig. 6. Speed-up with ray tracer on Kerrighed

We can observe in figure 6 and in figure 7 that the image size has little impact on speed-up. The best speed-up achieved is 11.77 (Kerrighed) respectively 11.88 (Plurix) on 12 nodes, which is fairly good result. The network is not the bottleneck for the ray tracer application because there is much more calculation than communication.

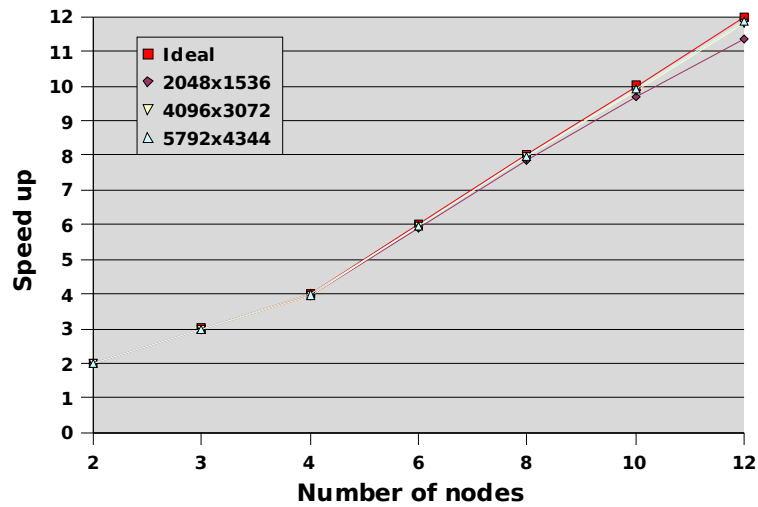


Fig. 7. Speed-up with ray tracer on Plurix

Furthermore, as there is no need for barrier-synchronization between steps of calculation, Plurix can fully utilize its optimistic synchronization model within the transactional consistency, because even the first write access to a page does not require the affirmation of all other nodes.

#### 4.6 Comparison of Kerrighed and Plurix

Kerrighed and Plurix are completely different systems: Kerrighed uses Linux and NFS to share files, whilst Plurix is an operating system from scratch without the need for any file system. Both Kerrighed and Plurix use page-based distributed shared memory, but Kerrighed uses an extended MESI protocol to exchange pages, whereas Plurix always transfers the most recently committed version of a page and uses the semantic group of a transaction to invalidate multiple pages.

Nevertheless, both systems perform almost linear with the ray tracer up to 12 nodes and show mature communication models, which are hidden from the application programmer completely, even though system knowledge will of course help in writing well performing applications as in any.

The SOR-algorithm uses high-volume communication in comparison to the time needed for calculation, as a consequence Fast Ethernet becomes the bottleneck.

### 5 Related Work

Numerous DSM projects have implemented a global memory management service on top of an existing OS, e.g. Solaris, Linux, and Windows NT. IVY was the first page-based DSM implementation (sequential consistency) followed by others implementations with weaker consistency models, e.g. TreadMarks (lazy release consistency), [9], [11]. To the researcher and to the students from these user-level systems provide important insight, especially about the relative merits of different consistency models [3]. But this approach introduces many programming constraints and limits performance. Specific run-time functions might be called by the programmer to data in the DSM or special storage classes might be defined.

The Single System Image idea has also been addressed by several projects in the past. The Sprite OS for example is written from scratch and provides a distributed file system and a process migration facility [12]. But Sprite does not allow to migrate threads and does not implement a global memory management mechanism.

The Mosix project extends Linux with a kernel-level process-migration facility. However, it does not provide any data sharing mechanism. Thus, processes can not share memory and threads can not be migrated in Mosix [13].

Plurix is the first OS tailored to a transactional DSM. Furthermore, there is no other system utilizing the DSM heap to distribute both data and code. Transactional consistency in the context of distributed computing is also proposed in [11]. The ideas discussed are similar to the transactional consistency in Plurix but the authors simulate a new CPU design for SMP machines rather than a cluster implementation.

### 6 Conclusions

The comparison of sequential and transactional consistency in Kerrighed and Plurix respectively shows that both perform adequately in spite of their strong consistency models. Efficiency is ensured in both systems by implementing the DSM at the kernel level and by avoiding the overhead of expensive context switches. Furthermore, Plurix benefits from the fact that several write operations are bundled into one transaction.

The SOR measurements revealed noticeable costs for the barrier synchronization in Plurix caused by a currently not optimal barrier implementation. Nevertheless, it is encouraging that a DSM organized as a heap storing code and data (Plurix) can compete with a traditional DSM approach (Kerrighed) allocating only dedicated data in DSM. Experiments with faster networks like Gigabit Ethernet, Myrinet, and Infiniband are planed in future work.

This work has been funded by the German DAAD within the PROCOPE program and the French ministry of foreign affairs. The work on Kerrighed has been partly financed by the Direction Générale de l'Armement (DGA) for the COCA project and by EDF R&D.

## 7 References

1. [www.kerrighed.org](http://www.kerrighed.org)
2. [www.plurix.de](http://www.plurix.de)
3. D. Mosberger, "Memory Consistency Models", *ACM Operating Systems Review*, 27(1), 18-26, January 1993.
4. R. Goeckelmann, S. Frenz, M. Schoettner, P. Schulthess, "Compiler Support for Reference Tracking in a type-safe DSM", Proceedings of the Joint Modular Languages Conference Klagenfurt, Austria, 2003.
5. M. Wende, M. Schoettner, R. Goeckelmann, T. Bindhammer, P. Schulthess, "Optimistic Synchronization and Transactional Consistency", Proceedings of the 2nd IEEE International Symposium on Cluster Computing and the Grid, Berlin, Germany, 2002.
6. N. Wirth and J. Gutknecht, *Project Oberon - The Design of an Operating System and Compiler*, Addison-Wesley, 1992.
7. Renaud Lottiaux and Christine Morin, "Containers: A Sound Basis For a True Single System Image", in: Proceeding of IEEE International Symposium on Cluster Computing and the Grid (CCGrid '01), Brisbane, Australia, 2001.
8. Geoffroy Vallée, Renaud Lottiaux, Louis Rilling, Jean-Yves Berthou, Ivan Dutka-Malhen, and Christine Morin, "A Case for Single System Image Cluster Operating Systems: Kerrighed Approach", *Parallel Processing Letters*, 13(2), June 2003.
9. K. Li., "IVY: A Shared Virutal Memory System for Parallel Computing", in: Proceedings of the International Conference on Parallel Processing, 1988.
10. P. Keleher et al., "TreadMarks: Distributed Shared Memory on Standard Workstations and Operating Systems", in: USENIX Winter 1994, 1994.
11. Lance Hammond, Brian D. Carlstrom, Vicky Wong, Ben Hertzberg, Mike Chen, Christos Kozyrakis, and Kunle Olukotun, "Programming with Transactional Coherence and Consistency", in: Proceedings of the International Conference on Architectural Support for Programming Languages and Operating Systems, Boston, USA, 2004.
12. John Ousterhout, A. Cherenson, Fred Dougliis, M. Nelson, Brent Welch, "The Sprite network operating system", *Computer*, 21(2):23-36, February, 1988.
13. Amnon Barak, S. Geday, Richard Wheeler, "The MOSIX Distributed Operating System", volume 672 of Lecture Notes in Computer Science, Springer, 1993.
14. <http://graphics.csail.mit.edu/classes/6.837/F01/Project05/project5.html>