3. Operating Systems

Steve Chapin, Syracuse University, USA and Joachim Worringen, RWTH Aachen, University of Technology, Germany

3.1. Introduction

Just as in a conventional desktop system, the operating system for a cluster lies at the heart of every node. Whether the user is opening files, sending messages, or starting additional processes, the operating system is omnipresent. While users may choose to use differing programming paradigms or middleware layers, the operating system is almost always the same for all users.

A generic design sketch of an operating system is given in Figure 3.1. It shows the major building blocks of a typical cluster node with the hardware resources located at the bottom, a monolithic kernel as the core of the OS, and system and user processes as well as some middleware extensions running on top of the kernel in the user space.

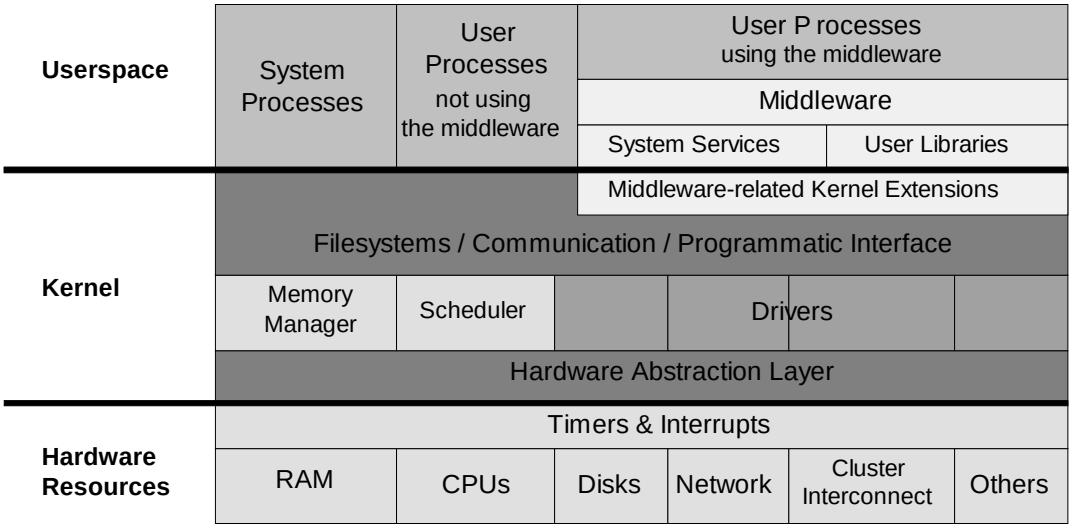

Figure 3.1. Generic OS design sketch for a typical cluster node

Some OS feature a distinct layer to abstract the hardware for the kernel. Porting such an OS to a different hardware requires only the need to adapt this abstraction layer. On top of this, the core functionalities of the kernel execute: memory manager, process and thread scheduler and device drivers to name the most important. Their services in turn are offered to the user processes by file systems, communication interfaces and a general programmatic interface to access kernel functionality in a secure and protected manner.

User (application) processes run in user space, along with system processes called daemons. Especially in clusters, the user processes often not only use the kernel functions, but also utilize additional functionality that is offered by middleware. This functionality is usually located in user libraries, often supported by additional system services (daemons). Some middleware extensions require extended kernel functionality, which is usually achieved by loading special drivers or modules into the kernel.

What then, is the role of the operating system in a cluster? The primary role is the same twofold task as in a desktop system: multiplex multiple user processes onto a single set of

hardware components (resource management), and provide useful abstractions for high-level software (beautification). Some of these abstractions include protection boundaries, process/thread coordination and communication as well as device handling. Therefore, in the remainder of this section, we will examine the abstractions provided by current cluster operating systems, and explore current research issues for clusters.

3.2. Background and Overview

The ideal operating system would always help, and never hinder, the user. That is, it would help the user (which in this case is an application or middleware-designer) to configure the system for optimal program execution by supplying a consistent and well-targeted set of functions offering as many system resources as possible. After setting up the environment, it is desired to stay out of the user's way avoiding any time-consuming context switches or excessive set up of data structures for performance-sensitive applications. The most common example of this is in high-speed message passing, in which the operating system pins message buffers in DMA-able memory, and then allows the network interface card, possibly guided by user-level message calls, to transfer messages directly to RAM without operating system intervention. On the other hand, it might be desired that the operating system offer a rich functionality for security, fault-tolerance and communication facilities – which of course contrasts with the need for performance that is omnipresent.

Exactly what attributes a cluster operating system should possess is still an open question. Here is a small list of desirable features:

- *Manageability*: An absolute necessity is remote and intuitive system administration; this is often associated with a Single System Image (SSI) which can be realized on different levels, ranging from a high-level set of special scripts, perhaps controlled via Java-enabled graphical front-end, down to real state-sharing on the OS level.
- *Stability*: The most important characteristics are robustness against crashing processes, failure recovery by dynamic reconfiguration, and usability under heavy load.
- *Performance*: The performance critical parts of the OS, such as memory management, process and thread scheduler, file I/O and communication protocols should work in as efficiently as possible. The user and programmer should be able to transparently modify the relevant parameters to fine-tune the OS for his specific demands.
- *Extensibility*: The OS should allow the easy integration of cluster-specific extensions, which will most likely be related to the inter-node cooperation. This implies, at a minimum, user-loadable device drivers and profound documentation of interfaces in kernel- as well as in user-space to allow for the development of specific extensions. The best way to provide extensibility is probably the provision of the source code because it reveals all interfaces and allows for modification of existing functionality (instead of having to design replacement parts from scratch). A good example for this is the MOSIX system that is based on Linux (see chapter 4).
- *Scalability*: The scalability of a cluster is mainly influenced by the provision of the contained nodes, which is dominated by the performance characteristics of the interconnect. This includes the support of the OS to be able to use the potential performance of the interconnect by enabling low-overhead calls to access the interconnect (inter-node scalability). However, clusters are usually built with SMP nodes with an increasing number of CPUs contained in each node. The ability of the OS to benefit from these is determined by its intra-node scalability. The intra-node scalibility is dominated by the process and thread schedulers and by the degree of parallelism in the kernel that the OS allows. It also includes the resource limits that an OS inhibits, foremost the maximum size of usable address space.

- *Support*: Many intelligent and technically superior approaches in computing failed due to the lack of support in its various aspects: which tools, hardware drivers and middleware environments are available. This support depends mainly on the number of users of a certain system, which in the context of clusters is mainly influenced by the hardware costs (because usually dozens of nodes are to be installed). Additionally, support for interconnect hardware; availability of open interfaces or even open source; support or at least demand by the industry to fund and motivate research and development are important. All this leads to a user community that employs required middleware, environments and tools to, at least, enable cluster applications.
- *Heterogeneity*: Clusters provide a dynamic and evolving environment in that they can be extended or updated with standard hardware just as the user needs to or can afford. Therefore, a cluster environment does not necessarily consist of homogenous hardware. This requires that the same OS should run across multiple architectures. If such an OS dose not exist for the given hardware setup, different OS need to be used. To ensure the portability of applications, a set of standardized APIs (like [1]) should be supported by the different operating systems. The same is true for the required middleware layers, which are frequently used to enable a cluster for heterogeneous use.

It should be noted that experience shows that these goals may be mutually exclusive. For example, supplying a SSI at the operating system level, while a definite boon in terms of manageability, drastically inhibits scalability. Another example is the availability of the source code in conjunction with the possibility to extend (and thus modify) the operating system on this base. This property has a negative influence on the stability and manageability of the system: over time, many variants of the operating system will develop, and the different extensions may conflict when there is no single supplier.

3.3. Technological Scope

In this sub-section, we will touch on some of the aspects of operating systems for clusters that frequently lead to discussions among the developers and users in this area. Some of these aspects are covered in more detail in other sections of this paper.

3.3.1 Functionality: OS versus Middleware

There has been little work on operating systems specifically for clusters. Much of the work one might consider as affecting the operating system, e.g. the GLUnix [2] work at U.C. Berkeley, is actually middleware. There are good reasons for this. First, operating systems are complex, and it is not always clear how a proposed change will affect the rest of the system. Isolating these changes in middleware can make good sense. Second, the applicability of the new functionality added by such a middleware layer is usually not limited to a single operating system, but can be ported to other operating systems as well.

For example, support for Distributed Shared Memory (DSM) arises from shared address spaces, which in a conventional kernel would be inside the operating system. However, in distributed computing, DSM software such as SVMlib [3] is most often implemented in middleware, running on diverse operating systems such as Windows NT and Solaris. The advantage here is that a single middleware layer can provide services to multiple operating systems, without requiring access or changes to OS source code.

3.3.2 Single System Image (SSI)

Regarding the frequently desired feature of SSI, at least two variants of it should be distinguished: SSI for system administration or job scheduling purposes and SSI on a system-call level. The first is usually achieved by middleware, running daemons or services on each node delivering the required information to the administration tool or job scheduler. The latter would have to offer features like transparent use of devices located on remote nodes or using distributed storage facilities as one single standard file system. These features require extensions to current single-node operating systems.

The goal of these extensions is the cluster-wide transparent sharing of resources. Next to the sharing of I/O space via traditional network file systems or other, more sophisticated means of real parallel I/O, the sharing of RAM (Distributed Shared Memory, DSM) is an area in which a lot of research has been done. However, the usual approach to DSM is meant as a parallel programming paradigm to use shared memory between the processes of a parallel application distributed across a number of nodes. Therefore, it is mostly realized via user-level libraries [3] and not as a service offered by the OS. Integration of DSM into the OS has rarely been done [4], [6]. The reason for this is that the performance for general-purpose use (requiring strict sequential consistency) is often to low. Using relaxed consistency models improves performance, but requires that special care is taken by the user of the DSM system that prohibits offering it as a standard service by the OS. Next to using memory situated on remote nodes for DSM, some experiments [7] have been done to use it as OS-managed remote memory: current interconnect technologies such as SCI or Myrinet offer lower latencies and higher bandwidth of inter-node communication than what can be achieved between the primary storage level (RAM) and the secondary storage level (hard disk) in intra-node communication. This leads to the idea to use the memory of remote nodes instead of the local hard disk for purposes like swapping or paging. This is a promising approach, which is, however, limited by the fact that it requires permanent memory related load-imbalances inside the cluster that is not desired in environments of dedicated compute clusters.

An operating-system-level SSI implies detailed state sharing across all nodes of the cluster, and to this point, OS researchers and practitioners have been unable to scale this to clusters of significant size (more than a hundred nodes) using commodity interconnects. That does not mean that an OS-level SSI is a bad thing; for the vast majority of clusters, which have less than 32 nodes, an operating-system-level single-system image may be quite workable.

One can view Intel's Paragon OS as a cluster operating system, although the Paragon is not a cluster in the sense of this paper because it is not made of commodity components. Nevertheless, its design has much in common with todays clusters: independent nodes, stripped down to basic hardware functionality, running their own instances of an OS and communicating via a high-speed interconnect. The integration of the OS on the nodes into one SSI is something many administrators of today's clusters would like to see.

Sun's Solaris MC, on the other hand, is specifically intended for use in clusters as this paper addresses them. It provides some shared state between kernels (although much of the Solaris MC functionality is implemented at the user level). However, a weakness of current distributed operating systems is that state sharing is binary: they use all-or-nothing sharing, which inhibits scalability. The Tornado operating system [8] is designed for 2-level hierarchical clusters of workstations, although it is not yet clear how well this approach will work for more generalized clusters.

3.3.3 Heterogeneity

It is arguable whether one should attempt to accommodate heterogeneous hardware at the operating system level within a single cluster. There are definite efficiencies to be gained from homogeneous clusters, and it may well make economic sense to replace an existing cluster rather than doing incremental heterogeneous expansion. Even if one accepts heterogeneity as inevitable, the operating system may not be the best place to address it. What we really want is that, at some layer, we provide a homogeneous set of abstractions to higher layers.

The lowest level on which heterogeneity causes problems is the data representation – big-endian vs. little-endian. If such systems are to be connected, the adaptation of the different representations could also be done on the lowest level possible to gain suitable performance. However, approaches to do endian conversion in hardware have not yet been done (the closest thing we are aware of is the ability of the old MIPS processors to run in either endian-ness, although there was no dynamic conversion).

On the other hand, this can just as easily be at the middleware layer instead of at the operating system layer (the “end-to-end” argument in networking would argue for pushing this to the highest layer possible). Middleware systems have done an excellent job of providing the illusion of homogeneity in heterogeneous systems – consider the success of Java and the Java Virtual Machine. However, creating illusions does cost in terms of performance. Therefore, we consider an operating system that runs on heterogeneous hardware as more of a serendipitous benefit rather than a requirement.

3.3.4 User-level communication facilities

A key characteristic of high-performance clusters is the interconnect between the nodes normally not being an Ethernet-based network, but more sophisticated technologies like SCI [8], Myrinet [10], GigaNet [11], or other, mostly proprietary solutions. This kind of hardware offers communication bandwidth in the range of several Gbps, however, to make best use of this performance, it is required that the access to the interconnect adapter involves as little overhead as possible. Therefore, the involvement of the operating system in this kind of communication is not desired; everything is to be done in the user space, preferably by techniques like protected user-level DMA. However, it is expected that the multiplexing of an arbitrary number of processes to a single interconnect adapter is still possible in a secure manner. This imposes a difficult task to the developers of the interconnect adapter and its driver, and also to the designers of the operating system into which the driver is to be embedded. The VIA industry standard [12], as discussed in the communications section of this document, appears to be the future for low-latency, high-bandwidth cluster communication.

Approaches like VIA (and U-Net [13], an earlier design on which many parts of VIA are based) try to minimize the involvement of the OS into inter-process communication by moving as much functionality as possible from the kernel space into user space. This means the buffer and queue management for send and receive operations is done by the user process in user-space. The OS is only involved into the setup of communication channels, but no longer in the communication (depending on the network adapters capabilities). This increases performance by reducing the number of context switches and local buffer copy operations. The most radical form of this low-overhead communication is SCI, which maps memory of remote processes into the address space of the local process, reducing the inter-process, inter-node communication to simple load and store operations.

3.3.5 Optimized parallel I/O

Less work has been done on high-speed file input/output concepts than for high-speed inter-process communication. However, file I/O is crucial for the performance of many types of applications, scientific codes as well as databases. The usual way is to employ a number (one or more) of dedicated I/O nodes in the cluster. However, every shared resource represents a potential bottleneck in a system that has to be scalable. A good example for the problems involved in this is, once again, the Intel Paragon. Its I/O concept was used for traditional file input/output as well as for swapping, is organized in a complex, tree-oriented parallel manner and nevertheless did not deliver optimal performance. Clusters should follow other concepts by doing as much node-local I/O as possible to reduce inter-node communication and I/O contention, while maintaining a consistent global view of the I/O space.

Examples of this approach are PVFS [14] and PPFS [15] in which a flexible number of servers in a cluster provide general I/O services in a distributed manner. For specific I/O needs of scientific applications, specialized middleware libraries like PANDA [16], which supports efficient parallel I/O for huge multidimensional arrays, have been developed.

3.4. Current State-of-the-art

State-of-the-art can be interpreted to mean the most common solution as well as the best solution using today's technology. By far the most common solution current clusters is running a conventional operating system, with little or no special modification. This operating system is usually a Unix derivative, although NT clusters are becoming more common. We do not intend "cluster" to be synonymous with "cluster of PCs," although this is, again, the most common case, for reasons of economics.

The single most popular cluster operating system is Linux [26]. This is primarily for three reasons:

1. It is free,
2. It is an open source operating system, meaning that one is free to customize the kernel to ones liking. To date, this has usually meant specialized drivers for underlying high-speed networks, and other I/O optimizations.
3. For historic reasons: Don Becker selected Linux for the original Beowulf cluster (this was for both technical and social reasons, as the Linux kernel was free of any possible licensing problems, unlike the BSD derivatives at the time), and thus Beowulf-derived systems have also used Linux.

Beowulf [17] is not a single ready-to-run package for Linux clustering, but merely a collection of tools, middleware libraries, network drivers and kernel extensions to enable a cluster for specific HPC purposes. This projects includes valuable components and is another example of how an (open source) OS can be extended and adapted to specific needs.

Sun Microsystems has developed a multi-computer version of Solaris; aptly named Solaris MC [18] Solaris MC consists of a small set of kernel extensions and a middleware library. Solaris MC incorporates some of the research advances from Sun's Spring operating system, including an object-oriented methodology and the use of CORBA IDL in the kernel. Solaris MC provides a SSI to the level of the device, i.e. processes running on one node can access remote devices as if they were local. The SSI also extends to a global file system and a global process space.

The Puma operating system [19], from Sandia National Labs and the University of New Mexico, represents the ideological opposite of Solaris MC. Puma takes a true minimalist approach: there is no sharing between nodes, and there is not even a file system or demand-paged virtual memory. This is because Puma runs on the "compute partition" of the Intel Paragon and Tflops/s machines, while a full-featured OS (e.g. Intel's TflopsOS or Linux) runs on the Service and I/O partitions. The compute partition is focused on high-speed computation, and Puma supplies low-latency, high-bandwidth communication through its Portals mechanism.

MOSIX [20],[21] is a set of kernel extensions for Linux that provides support for seamless process migration. Under MOSIX, a user can launch jobs on their home node, and the system will automatically load balance the cluster and migrate the jobs to lightly-loaded nodes. MOSIX maintains a single process space, so the user can still track the status of their migrated jobs. MOSIX offers a number of different modes in which available nodes form a cluster, submit and migrate jobs, ranging from a closed batch controlled system to a open network-of-workstation like configuration. MOSIX is a mature system, growing out of the MOS project and having been implemented for seven different operating systems/architectures. MOSIX is free and is distributed under the GNU Public License.

Next to the cluster operating systems mainly used in research and for running computationally intensive applications (which do not require a high degree of OS-support), clustering is also in use in the commercial arena. The main commercial applications in this area are data-intensive and often involve database management systems (DBMS). AIX from IBM has a strong position here, running the SP family of cluster-based servers, featuring proven stability, good manageability and a number of clustered databases. From the other commercial Unix variants, Sun's Solaris has a strong focus on clustering, high availability (HA) and is also widely used in research. IRIX from SGI relies a sophisticated NUMA-SMP technology that provides a very specific kind of clustering. Other operating systems with an emphasis on HA (Tandem Non-Stop Unix and Unixware 7) are covered in the related section of this paper.

In the area of high-performance I/O, only a very limited number of proven and usable solutions exist. One of these is GPFS (General Parallel File System) [22] by IBM, specifically designed for the SP server running AIX. It is based on storage arrays (disks) being connected to one or more servers that exchange data with the clients via multithreaded daemons. It fulfils the criterion "usable" in that it features a kernel extension that can be accessed like any standard UNIX file system. This avoids the necessity of recompiling or even modifying applications that are to use GPFS. However, it performs best for large sequential reads and writes due to its technique of striping the data across the disks. Many technically more sophisticated but less general (and thus mostly less usable in a general cluster environment) research projects exist [23],[24] which often deal with the special, but frequently occurring scenario of collective I/O in large scientific applications.

We present a comparison of the most relevant operating systems used for clustering in Appendix A. We are aware that already the selection criteria "most relevant", and also the comparison of the selected systems will lead to discussions. However, covering every aspect of each system is surely beyond the scope of this paper, and a stimulation of the discussion is a desired effect. Apart from our own experiences, we consider the studies that have been performed by D.H. Brown Associates [25],[26].

3.5. Future work

As SMP become more common in clusters, we will see a natural hierarchy arise. SMP have tight coupling, and will be joined into clusters via low-latency high-bandwidth interconnection networks. Indeed, we fully expect that heterogeneous clusters of SMP will arise, having single, dual, quad, and octo-processor boxes in the same cluster. These clusters will, in turn, be joined by gigabit-speed wide-area networks, which will differ from SAN primarily in their latency characteristics. This environment will naturally have a three-level hierarchy, with each level having an order of magnitude difference in latency relative to the next layer.

This structure will have its greatest impact on scheduling, both in terms of task placement and in terms of selecting a process to run from a ready queue. Scheduling is traditionally considered an operating system activity, yet it is quite likely that at least some of this work will be carried out in middleware.

For example, as one of our research projects we are investigating thread management for mixed-mode (multi-threaded and message passing) computing using OpenMP and MPI, which we believe is a natural by product of clusters of SMP. Most cluster applications, particularly scientific computations traditionally solved via spatial decomposition, consist of multiple cooperating tasks. During the course of the computation, hot spots will arise, and a self-adapting program might wish to manage the number of active threads it has in each process. Depending on the relationship of new threads to existing threads (and their communication pattern) and the system state, a decision might be made to do one of the following:

- Add a new thread on an idle processor of the SMP where the process is already running.
- Expand the address space via distributed shared memory to include an additional node, and add a thread there.
- Add a thread to a non-idle processor already assigned to the process.
- Migrate the process to a larger SMP (e.g. from 2 nodes to 4 nodes) with an idle processor, and add a new thread there.

The described technique of thread placing and process migration is also related to the high-availability issue that is critical for commercial applications. Automatic and fast migration of running processes, taking benefit from advanced interconnect technology offering e.g. transparent remote memory access, will lift the definition of fail-over times into new dimensions.

We might also examine the issue of configurability. Users might want to alter the personality of the local operating system, e.g. "strip down" to a Puma-like minimalist kernel to maximize the available physical memory and remove undesired functionality. Possible mechanisms to achieve this range from a reload of a new kernel and a reboot to dynamically linking/unlinking code into/out of the kernel. This leads to the question: “How much (and which) functionality does a cluster operating-system need?” The more functionality a system has, the more complicated it gets to maintain and the greater the chance for malfunctions due to bad configurations or errors in the interaction of different parts of the system. Again, Linux is the easiest way for the majority of researchers to study this area.

Another important aspect of cluster computing in which the operating system is strongly involved is distributed file I/O. Current solutions of I/O systems are mostly static, do not adapt very well to the actual workload and thus tend to have bottlenecks, mostly by the limited number of dedicated I/O nodes to which all data has to be routed. The transparent filesystem-level support of distributed and adaptive I/O is an open issue for cluster operating systems. As an example for an attempt, we are currently implementing an MPI-IO implementation that operates on the basis of an existing DSM library on top of an SCI interconnect. This technique may result in a dedicated filesystem for high-bandwidth, low-latency requirements that is totally distributed among the participating clients.

3.6. Conclusions

Cluster operating systems are similar in many ways to conventional workstation operating systems. How different one chooses to make the operating system depends on ones view of clustering. On the one hand, we have those who argue that each node of a cluster *must* contain a full-featured operating system such as Unix, with all the positives and negatives that implies. At the other extreme, we see researchers asking the question, "Just how much can I remove from the OS and have it still be useful?" These systems are typified by the work going on in the Computational Plant project at Sandia National Laboratories. Still others are examining the possibility of on-the-fly adaptation of the OS layer, reconfiguring the available services through dynamic loading of code into the cluster operating system.

It is worth noting that every notable attempt to provide SSI at the OS layer has been regarded as a failure on some level. Sun's Solaris MC has never been offered as a product, and recently sun has approached academic computer scientists to evaluate Linux on Sparc stations as the basis of a cluster product. Intel's Paragon OS is well known for its tendency to lock up the entire machine because of minor problems on one node, as well as its wretched performance on large systems. We are not saying it is impossible to build a scalable SSI at the OS level, we are just saying that no one has done it, and we think there is a good reason. The forthcoming SIO standard will blur the edges between remote and local devices, and perhaps this will lead to more highly scalable SSI.

A final note is that, through a combination of OS improvements and acquisition of relevant technology, Microsoft has become a viable option in the realm of clustering. The HPVM [27] effort has demonstrated that it is possible to build a reasonable cluster from Windows NT boxes, and with the recent installation of clusters such as the one at Cornell University, NT or Windows 2000 is going to be a factor in the cluster OS picture for the foreseeable future.

To sum it up, clusters for technical and scientific computing based on Linux and other standalone Unix platforms like AIX are here, and they work. In the area of commercial cluster computing, Linux still lacks essential functionalities which “conventional” Unix systems and in parts even Windows NT do offer. It is to be observed if the free and distributed development model of Linux will be able to offer proven solutions in this area, too, since these topics are rarely addressed in the Linux developer community. Nevertheless, more exotic OS technology is and will be the current focus of many research efforts, both academic and commercial. There will probably never be “THE” cluster OS, as Linux will adopt research results much more quickly than commercial vendors, particularly Microsoft, if history is a reliable guide.

3.7. References

[1] IEEE Std. 1003.1: Information Technology-Portable Operating System Interface (POSIX)-Part 1: System Application: Program Interface (API) [C Language]. Institute of Electrical and Electronics Engineers, Inc., 1990.

[2] Global Layer Unix, http://now.cs.berkeley.edu/Glunix/glunix.html

[3] K. Scholtyssik and M. Dormanns, Simplifying the use of SCI shared memory by using software SVM techniques, *2nd Workshop Cluster-Computing*, Published in: W. Rehm, and T. Ungerer (Eds.), Cluster-Computing, Proc. 2. Workshop, 25-26. March 1999, Universität Karlsruhe (CSR-99-02), http://www.tu-chemnitz.de/informatik/RA/CC99/

[4] C. Amza, A.L. Cox, S. Dwarkadas, P. Keleher, H. Lu, R Rajamony, W. Yu and W. Zwaenepoel, TreadMarks: Shared Memory Computing on Networks of Workstations, *IEEE Computer*, Vol. 29, No. 2, 1996, http://standards.ieee.org/regauth/oui/tutorials/sci.html

[5] S. Zeisset, S. Tritscher, M. Mairandres, A New Approach to Distributed Memory Management in the Mach Microkernel, *USENIX 1996 Technical Conference*, San Diego, California, 1996

[6] G. Cabillic, T. Priol, I. Puaut, MYOAN: an implementation of the KOAN shared virtual memory on the Intel paragon, *Technical Report RR-2258*, Institute National de Recherche en Informatique et en Automatique, 1994

[7] E. Anderson, A. Mainwaring, J. Neefe, C. Yoshikawa, T. Anderson, D. Culler, D. Patterson, Experience with Two Implementations of Network RAM, *Internal Report, Computer Science Division*, University of California at Berkeley, http://http.cs.berkeley.edu/~eanders/projects/netram/

[8] B. Gamsa and O. Krieger and J. Appavoo and M. Stumm, Tornado: Maximizing Locality and Concurrency in a Shared Memory Multiprocessor Operating System, In the Proceedings of the 3rd Symposium on Operating Systems Design and Implementation (OSDI), pp. 87-100, February 1999.

[9] SCI Association, http://www.SCIzzL.com; SCI Products – http://www.dolphinics.com

[10] Myrinet, http://www.myri.com

[11] GigaNet, http://www.giganet.com

[12] VIA, http://www.viarch.org

[13] U-Net, http://www.cs.berkeley.edu/~mdw/projects/unet; M. Welsh, A. Basu, and T. von Eicken: ATM and Fast Ethernet Network Interfaces for User-Level Communication, Proc. of *High-Performance Computer Architecture 3*, San Antonio, February 1997.

[14] PVFS, http://parlweb.parl.clemson.edu/pvfs

[15] PPFS, http://www.crpc.rice.edu/CRPC/newsletters/win97/work_PPFS.html

[16] PANDA, http://drl.cs.uiuc.edu/panda

[17] Beowulf, http://www.beowulf.org

[18] Solaris MC, http://www.sunlabs.com/research/solaris-mc

[19] Puma, http://www.cs.sandia.gov/puma

[20] MOSIX, http://www.mosix.org

[21] A. Barak, O. La'adan and A. Shiloh, Scalable Cluster Computing with MOSIX for LINUX (ftp), *Proc. Linux Expo '99*, pp. 95-100, Raleigh, N.C., May 1999.

[22] J. Barkes, M.R. Barrios, F. Cougard, P.G. Crumley, D. Martin, H. Reddy, and T. Thitayanum, GPFS: A Parallel File System, IBM *International Technical Support Organization*, SG24-5165-00, April 1998

[23] D. Kotz, Parallel I/O Archive, Dartmouth College, http://www.cs.dartmouth.edu/pario/

[24] Yong E. Cho, Efficient Resource Utilization for Parallel I/O in Cluster Environments, *Ph.D Thesis, University of Illinois* at Urbana-Champaign, 1999
[25] *1998-1999 Operating System Function Review*, D.H. Brown Associates Inc., http://www.rs6000.ibm.com/resource/consult/dhbrown/osfrev.html, 1998
[26] *Linux – How Good Is It?* Executive Summary, D.H. Brown Associates Inc., http://www.dhbrown.com, 1999
[27] HPVM, http://www-csag.ucsd.edu/projects/hpvm.html