Perspectives for High Performance Computing in Workstation Networks

Volker Strumpen, Balkrishna Ramkumar,
Thomas L. Casavant and Sudhakar M. Reddy

Department of Electrical and Computer Engineering,
University of Iowa, Iowa City, Iowa 52242
email: {strumpen,ramkumar,tomc,reddy}@eng.uiowa.edu

Abstract. Networks of workstations have become increasingly popular for high performance computing. However, in order to become a real alternative for MPPs, reliability and efficiency issues must be tackled. In this paper, we identify the key challenges for very large workstation networks, and describe implementation techniques at system software level to overcome these problems. CROWN, a testbed for experimenting with these mechanisms is briefly discussed.

1 Introduction

High Performance Computing (HPC) to date has been attempted via one of three primary approaches: Vector-supercomputers (e.g., Cray), Massively Parallel Processors (MPPs, such as the Intel Paragon), and Networks of Workstations (NOWs). Vector supercomputers are expensive and MPPs are not cost-effective and often considered difficult to program. NOWs suffer from high latencies and scheduling overhead that is endemic in time-shared environments.

Figure 1 projects the growth of workstation performance over the next two decades and includes an estimate for the necessary computation power of a protein folding problem [27]. The most important mandate for HPC in the years to come will be to be able to address such applications for a wide community of users. This will require the following three system characteristics: (1) *Scalability*, with respect to workstation and network generations [3] as well as performance and system configuration; (2) *Accessibility* of the system to a geographically dispersed community of users; (3) *Affordability* of the system to this community of users. Supercomputers, MPPs, and current NOW-based systems do not effectively satisfy one or more of these requirements.

A *very large* and *geographically dispersed* network of high performance workstations such as the Internet provides the potential resources to solve these problems. For such a system to be viable and effective, two technical challenges have to be overcome:

Reliability: A very large geographically distributed system of workstations suffer from poor availability. Figure 2 illustrates the availability of workstations based on failure data gathered by Wood [34]. The dramatic loss of availability with growing numbers of processors prevents scalability.

Efficiency: The current Internet is flooded by information retrieval traffic and has become essentially unusable for HPC. Furthermore, the overheads and poor reliability of conventional operating systems are likely to hinder efficient HPC.

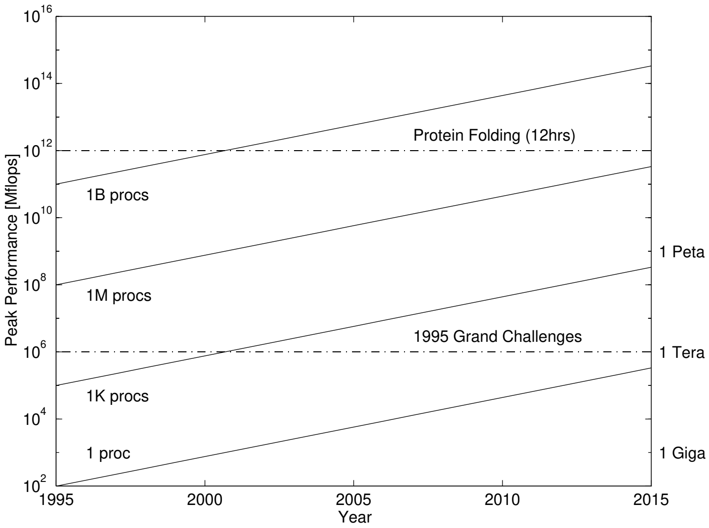

Fig. 1. Future performance development.

We argue that the context for addressing the two problems identified above is a *virtually dedicated* network of high performance workstations which runs an operating system with a shifted protection boundary. In contrast to skimming idle cycles, as done by most other projects such as NOW [1] or PVM [11], our notion of a virtually dedicated system is a single-user system that provides quantitative predictability of resources and increased reliability. To avoid complete physical dedication, virtual dedication provides weaker guarantees for quality of service. An operating system with the protection boundary shifted to the job submission interface will eliminate high overheads due to run-time integrity/protection checking by permitting applications to run in *trusted mode*. The critical research issues to be addressed in order to overcome these last two challenges center on the following:

1. Transparent reliability mechanisms for applications.
2. Scheduling issues related to the proposed distributed HPC environment.
3. Latency tolerance and hiding mechanisms.
4. Integrity checking of application programs.

A system that implements these issues will be *scalable, affordable, accessible, reliable, and efficiently usable*. We envision that future HPC systems will be organized in this manner.

2 Challenges for NOW-based HPC

A primary challenge of systems research in workstation environments is to provide users with access to a NOW-based high-quality HPC environment. We consider each of the problems encountered with this endeavor.

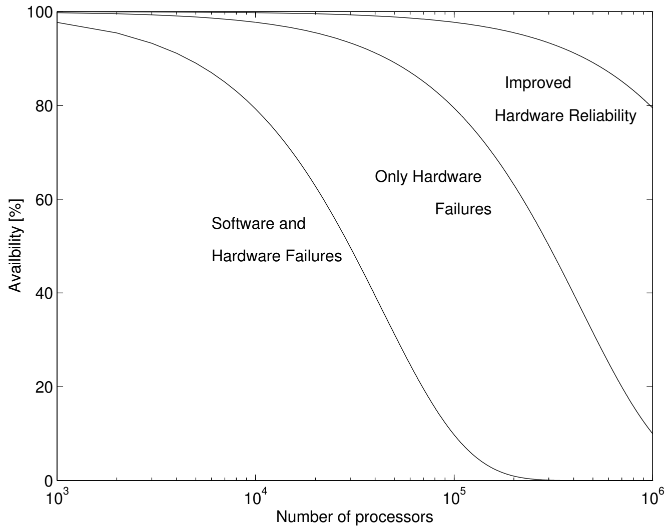

Fig. 2. Probability of availability.

Scalability. NOWs, in contrast to vector-supercomputers and MPPs, are scalable in principle. Commodity workstations and networks are generation scalable and size scalable. Vendors provide the necessary interfaces to connect new generations of workstations as well as new networks to the Internet. However, the reliability and efficiency issues affecting the Internet are the essential obstacles for scalability today.

Affordability. The exponential growth of the Internet further demonstrates the affordability of a very large system by dividing the investment among many implicitly cooperative parties. In contrast, only a few communities manage to afford vector-supercomputers, and even fewer can afford an MPP. However, the current Internet lacks an organizational concept that motivates owners of machines to achieve a viable working relationship by contributing their machines to a very large, geographically dispersed pool of workstations for *computation purposes*.

Accessibility. Whereas accessibility has been demonstrated in a large academic environment comprising about 1,000 machines [28], this issue is a major psychological and administrative problem in general. Only with a sound organization that ensures security, and is capable of motivating owners of machines to participate in a very large configuration, can this problem be solved.

Reliability. Usually, reliability is being viewed as an absolute characteristic of systems that must be delivered *at any cost*. Tandem systems [13, 17] are an example of this approach. Another notion of reliability is the probabilistic *performability* measure [22] which combines performance and dependability, and models the accomplishment (reward) made by the system at user-level. Up to now, fault tolerance issues have been treated extensively at the hardware and operating system level. Several subsystems of operating systems have been designed with respect to the *end-to-end argument* [26] that address fault tolerance issues at the operating system level. A large body of work exists on fault tolerance above the operating system level to enhance the reliability of parallel programs. Some efforts consider fault tolerance at all levels of system design, for example the *responsive systems* approach [21]. However, there is no agreement on how to provide application fault tolerance transparently in general.

Today's *availability* of workstations is not suited for large-scale network computing. Wood [34] analyzed the availability of a `typical' client/server architecture comprising 1,000 desktop machines. He found that over a one year period, a single machine was available on average 97.7 % of the time. Interpreting Wood's data as a probabilistic measure, the availability of workstation networks can be calculated as given in Figure 2. *Software and hardware failures* of today's systems lead to less than 10 % availability when coupling 100,000 machines. However, this situation can be improved. 90 % of the failures reported by Wood are software failures. If these could be eliminated the curve indicating *only hardware failures* would be obtained. To achieve an availability of 80 % with a network of one million machines, either *improved hardware reliability* without software failures, or a run-time environment that transparently recovers from the remaining software and hardware failures must be provided.

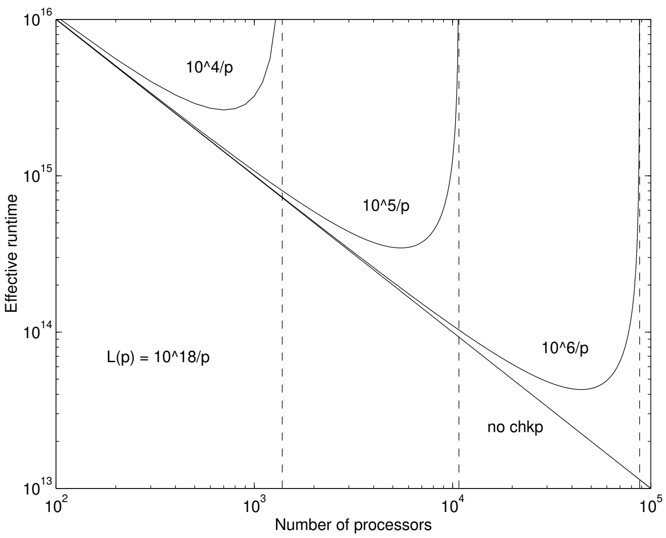

Fig. 3. Dependency of effective runtime on number of processors and number of checkpoints. The attributes at the poles give the time units between checkpointing.

Figure 3 illustrates the impact of checkpointing on the effective runtime of an application. The family of curves plots a model for the effective runtime with periodical checkpointing and without failures. To facilitate discussion, but without loss of generality, the idealized load of the parallel program for p processors is assumed to be

$L(p) = 10^{18}/p$, according to the protein folding application given in Figure 1. The overhead of a single checkpoint is $O_{ch} = \log(p)$, and the time between two successive checkpoints varies $T_{ch}(p) \in \{T_f/p, 0.1T_f/p, 0.01T_f/p\}$, where T_f is the mean time between failure. Since the probability of failures increases proportional to the number of processors used, we assume that a larger number of checkpoints is saved at smaller time intervals $T_{ch}(p)$. The curve attributed *no chkp* gives the effective runtime without checkpointing $E_{ch}(p) = L(p)$. The other three curves show the impacts of checkpointing and number of processors on the effective runtime: (1) There is a maximum number of processors for which $T_{ch}(p_{max}) = O_{ch}(p_{max})$ and thus $E_{ch}(p_{max}) = \infty$. When adding processors beyond p_{max} the computation does not progress, because the system is consumed with checkpointing activities. (2) The effective runtime has a minimum for a particular number of processors $p_{opt}(T_{ch})$. Approaching this minimum must be the goal of any scheduling strategy incorporating reliability.

Efficiency. Past experience has shown that skimming off idle cycles does not deliver high performance due to the scheduling overhead. The very large, dynamically and unpredictably varying latencies of the Internet, as well as the time-shared operation of open systems impede or even prevent the programming of efficient parallel code. Since the promotion of the World-Wide-Web and the Internet into the commercial world, the demands on networking services have been growing faster than the bandwidth of the Internet. Furthermore, the growing number of users and complexity of software used in the net imposes a serious threat on its reliability.

Providing efficient mechanisms for all kinds of operating system tasks, ranging from protection mechanisms to low latency communication and high bandwidth I/O, is the major goal of many operating system related efforts. A large amount of work concentrates on shifting functionalities out of the kernel into the application's non-privileged "user-level", ranging from communication protocols [7, 20, 31] to new customizable operating system design [4, 8, 19, 24]. Performance can also be improved by keeping the data path within the kernel and prevent it from crossing the kernel-user boundary to avoid unnecessary context switches. This is advantageous for applications that stream data from one device to another. Fall and Pasquale [9] report increased data transfer bandwidth as well as a higher availability of the CPU for other useful (interleaved) computation. The numbers reported are similar to the communication latency hiding capability of stock workstation hardware reported in [29]. The effects of reducing the copies in the communication protocol's data path are reported in [2, 6, 7, 23, 31, 32]. All these approaches have a common objective: Increasing performance while maintaining existing protection mechanisms or boundaries. We argue that these approaches tackle only the tip of the iceberg. Abandoning nearly all kernel protection will allow for substantial performance improvements.

Besides speeding up hardware access locally, system functionalities involving network access such as interprocess communication [30], file access, scheduling, replication of checkpoints and recovery need to be accelerated or hidden. Compared to an MPP, system latencies will remain relatively high in a geographically dispersed system that requires modularization and network switches or hubs which introduce substantial delays.

3 The CROWN Approach

In the CROWN project — Computing Reliably On Workstation Networks — we propose a virtually dedicated network of machines that provides a more predictable set of resources. We regard a guarantee for quantitative predictability of resources as essential for writing efficient parallel application programs. Ideally, we would gain exclusive access to both computers and the network, albeit for limited periods of time. Thus, instead of operating in a shared, open environment such as the Internet, we prefer a dedicated system in much the same way than the dedicated mode of supercomputers. We envision that future HPC systems for solving very large problems such as protein folding will be distributed systems, which are affordable by dividing the investment cost among many users and trading compute resources in much the same way as a commodity market provides investment return to private investors, income stability to producers, and products to consumers. Whereas such a market model will allow for dedicated access to single machines, there is no clear solution for the network yet. The Internet, which currently connects open systems is not dedicated, and the resources available to a single user are not predictable. Future networking technologies such as ATM that should provide guaranteed bandwidth to a virtual channel [10] are promising. If this bandwidth can be negotiated, it is straightforward to incorporate ATM networks into a market model that offer quality of service based on cost.

Once users are motivated to invest into the CROWN configuration and trust this system, we can setup a virtually dedicated system by operating individual machines in single-user mode, which renders multi-user protection superfluous. If owners dedicate their high performance equipment for a relatively long period of time, for example during night, we can afford switching to a single-user operating system and back to the original multi-user state when this period expires. A job scheduler is responsible for space partitioning and accounting for reconfiguration of the CROWN system. Assuming that a very large number of machines is available such that the average utilization of the system is relatively low, the single-user mode is sufficient. This assumption distinguishes a CROWN configuration from supercomputers, where high utilization is mandatory to amortize the investment.

Low Latency and Latency Tolerance. In the virtually dedicated operational mode, most protection mechanisms can be moved from the operating system kernel to the CROWN compiler, linker and scheduler. We will customize an existing operating system (UNIX) by removing all multi-user protections, instrumenting it with a new scheduler, communication and checkpointing modules, and will lift the majority of the operating system into the user space such that it forms a library which will be linked to the application. The new aspects of this design will be the integration of runtime support and compiler extensions for program specific checkpointing and recovery, and integrity checks for increased performability.

Figure 4 shows how the protection domain is shifted to the user interface, which is a new and key aspect of our investigation. The kernel boundary will be removed for the sake of performance. Instead, most protection and security checks will be done at the compiler, linker and scheduler of the CROWN system. Since the CROWN configuration is entirely transparent to the user, the *integrity check* involves programs submitted by the

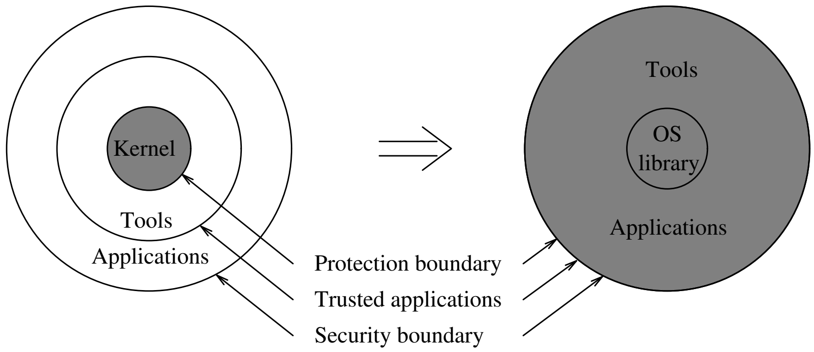

Fig. 4. Shift of protection boundary: From the kernel boundary to the CROWN system interface.

user. Within the CROWN system the service of proprietary programs such as compilers will enable these checks.

Transparent Reliability. Applications should be fault tolerant without special precautions taken by the programmer, or purchasing expensive hardware. The CROWN programming environment supports reliability transparently with the help of compiler and runtime support. The challenge is to reduce the overhead of checkpointing and recovery such that performability is improved. Optimizing the common case, compiler analysis can be used to determine suitable points in the program to checkpoint, thereby generating program specific code that minimizes the size of the checkpoints. This needs to be supplemented by fast timer-based runtime support for checkpointing when compiler analysis proves insufficient.

Job Scheduling. Conventional job schedulers in parallel machines run applications in space partitions, because time-sharing is too inefficient. The CROWN job scheduler plays the broader role of an *interface* between the user and the CROWN system. In contrast to conventional job scheduling, the user provides profile information to the scheduler that allows the scheduler to make decisions and, depending on the reliability of the configuration, estimates a price to be charged to the user. Once an acceptable bid/configuration has been agreed upon, the a trusted reliability compiler uses this information to instrument the submitted program for checkpointing and recovery. The scheduler is also responsible for selecting a migration target in case that an application cannot terminate on a workstation before the CROWN-dedicated time span expires.

Integrity Checking. The problem introduced by shifting the protection boundary is that of how to perform the integrity checking. For performance reasons, we would like to do all checks at compile time. However, this will not always be possible. We do not know yet, which method will be the best. Nevertheless, there are several options prone to investigation: Runtime checks might be necessary which could be based on techniques such as *software based fault isolation* [33]. Another possibility is to use traditional virtual memory technologies to retain firewalls at process address space boundaries.

4 Related Work

Previous research on NOWs has been driven by three observations: (1) The potential of available and inexpensive compute resources is tremendous, (2) these resources are hardly utilized, and (3) the Internet is an in-place means for compounding their power. The general idea arising from this situation has been to skim off the idle cycles from time-shared workstations, and make them available for parallel computing. This kind of *network computing* on workstations has gained considerable attention since the PVM [11] and MPI [25] projects have attracted many users. Similar projects tackling the system level of network computing include [1, 5, 12, 14, 15, 16, 18, 29]. The technological developments eventually leading to NOWs as a primary compute resource have been analyzed in [1, 29].

PVM [11] is the most popular message passing system to date. It provides a runtime library that is focused on a portable programmer interface for heterogeneous network computing. The user base of PVM is the scientific computing community. PVM is not intended as a general purpose computing environment, but as a parallel programmer interface. Therefore, the PVM project does not tackle the challenges of the underlying system. In fact, PVM or MPI [25] are popular candidates for a programmer interface of CROWN.

The Berkeley NOW project [1] focuses on providing a network of workstations as a general purpose computer. The reasoning for NOW is very similar to that of CROWN. However, the goals and philosophies are different. The philosophy of NOW does not rate parallel computing as a topic of primary importance. Instead, [1] states: "... the challenge is to make NOW a win for all users – it should deliver at least the interactive performance of a dedicated workstation while providing the aggregate resources of the network for demanding sequential and parallel programs." In contrast, CROWN is aiming primarily at high performance computing, which raises entirely different technical problems. Of course, users with less demanding applications will profit from such a system as well. Furthermore, NOW does not identify reliability as a major challenge, because it limits the scope to LAN-based clusters.

5 Conclusion

NOWs offer an inexpensive yet powerful medium for HPC. We discussed the technical key challenges of NOWs for large-scale network supercomputing: reliability and efficiency. Furthermore, we identified those research topics that need to be addressed in order to solve these technical problems: (1) Shifting the protection domain from the kernel to the user interface, performance enhancements are possible at system level that allow for low latency communication and efficient latency tolerance mechanisms; (2) the reliability issue will be tackled at the compiler and runtime level. Portable checkpoints for heterogeneous networks and compiler generated, optimized checkpointing code, as well as latency hiding for checkpointing are necessary in order to reduce the overhead of checkpointing; (3) Scheduling mechanisms for placing parallel applications, replication and migration are needed to optimize the response time of a parallel application. Furthermore, (4) the user interface has to provide tools for integrity checking of all application programs that are entered into the system.

References

1. Thomas E. Anderson, David E. Culler, David A. Patterson, and the NOW team. A case for NOW (networks of workstations). *IEEE Micro*, 15(1):54–64, February 1995. (URL `http://now.cs.berkeley.edu/Papers/case.ps`).
2. D. Banks and M. Prudence. A high-performance network architecture for a PA-RISC workstation. *IEEE Journal on Selected Areas in Communications*, 11(2):191–202, February 1993.
3. Gordon Bell. Ultracomputers — a teraflop before its time. *Communications of the ACM*, 35(8):27–47, August 1992.
4. Brian N. Bershad, Craig Chambers, Susan Eggers, Chris Maeda, Dylan McNamee, Przemyslaw Pardyak, Stefan Savage, and Emin Gün Sirer. SPIN - an extensible microkernel for application-specific operating system services. Technical Report UW-CSE-94-03-03, Department of Computer Science and Engineering, University of Washington, February 1994. (URL `ftp://ftp.cs.washington.edu/tr/1994/03/UW-CSE-94-03-03.PS.Z`).
5. Robert D. Blumofe, Christopher F. Joerg, Bradley C. Kuszmaul, Charles E. Leiserson, Keith H. Randall, and Yuli Zhou. Cilk: An efficient multithreaded runtime system. In *5th ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming*, pages 207–216, Santa Barbara, California, July 1995.
6. José C. Brustoloni and Brian N. Berschad. Simple protocol processing for high-bandwidth low-latency networking. Technical Report CMU-CS-93-132, School of Computer Science, Carnegie Mellon University, March 1992.
7. Peter Druschel and Larry L. Peterson. Fbufs: A high-bandwidth cross-domain transfer facility. In *14th ACM Symposium on Operating Systems Principles*, pages 189–202, Asheville, North Carolina, December 1993. (URL `file://ftp.cs.arizona.edu/xkernel/Papers/fbuf.ps`).
8. Dawson R. Engler, M. Frans Kaashoek, and James W. O'Toole, Jr. The operating system kernel as a secure programmable machine. To appear in Operating Systems Review, Special Issue on Extensible Operating Systems, 1995. (URL `http://www.pdos.lcs.mit.edu/ engler/xsigops.ps`).
9. Kevin Fall and Joseph Pasquale. Exploiting in-kernel data paths to improve I/O throughput and CPU availability. In *1993 Winter USENIX Conference*, pages 327–333, San Diego, California, January 1993. USENIX Association.
10. ATM Forum. *ATM User-Network Interface Specification*. Prentice-Hall, New Jersey, 1995.
11. A. Geist, A. Beguelin, J. Dongarra, W. Jiang, R. Manchek, and V. Sunderam. *PVM: Parallel Virtual Machine — A Users' Guide and Tutorial for Networked Parallel Computing*. MIT Press, Cambridge, 1994.
12. David Gelernter and David Kaminsky. Supercomputing out of recycled garbage: Preliminary experience with Piranha. In *1992 ACM International Conference on Supercomputing*, pages 417–427, Washington, D.C., July 1992.
13. Jim Gray. A census of tandem systems. *IEEE Transactions on Reliability*, 39(4):409–418, October 1990.
14. Andrew S. Grimshaw, William A. Wulf, James C. French, Alfred C. Weaver, and Paul F. Reynolds, Jr. Legion: The next logical step towards a nationwide virtual computer. Technical Report CS-94-21, Department of Computer Science, University of Virginia, June 1994.
15. Laxmikant V. Kale. The Chare kernel parallel programming language and system. In *International Conference on Parallel Processing*, volume II, pages 17–25. IEEE, August 1990.
16. H. T. Kung, R. Sansom, S. Schlick, P. A. Steenkiste, M. Arnould, F. J. Bitz, E. C. Cooper, O. Manzilcioglu, D. Ombres, and B. Zill. Network-based multicomputers: An emerging

parallel architecture. In *Supercomputing '91*, pages 664–673, Albuquerque, New Mexico, November 1991.
17. Inhwan Lee and Ravishankar K. Iyer. Faults, symptoms, and software fault tolerance in the tandem guardian90 operating system. In *23rd International Symposium on Fault-Tolerant Computing*, pages 20–29, 1993.
18. M. J. Litzkow, M. Livny, and M. W. Mutka. Condor - a hunter of idle workstations. In *8th International Conference on Distributed Computing Systems*, pages 104–111, San Jose, California, July 1988.
19. Steven Lucco. Bridge integrated multiserver project, 1995. (URL `http://www.cs.cmu.edu/afs/cs/project/sfi/www/top.html`).
20. Chris Maeda and Brian N. Bershad. Protocol service decomposition for high-performance networking. In *14th ACM Symposium on Operating Systems Principles*, pages 244–255, Asheville, North Carolina, December 1993.
21. Miroslaw Malek, Andreas Polze, and Matthias Werner. A framework for responsive parallel computing in network-based systems. In *International Workshop on Advanced Parallel Processing Technologies*, pages 335–343, Bejing, China, September 1995.
22. John F. Meyer. Performability: A retrospective and some pointers to the future. *Performance Evaluation*, 14(3–4):139–156, February 1992.
23. Ron Minnich, Dan Burns, and Frank Hady. The memory-integrated network interface. *IEEE Micro*, 15(1):11–20, February 1995.
24. Allen B. Montz, David Mosberger, Sean W. O'Malley, Larry L. Peterson, Todd A. Proebsting, and John H. Hartman. Scout: A communications-oriented operating system. Technical Report TR 94-20, Department of Computer Science, University of Arizona, June 1994.
25. MPI: A Message-Passing Interface Standard. Message Passing Interface Forum, April 1994. (Distribution: `netlib`).
26. J. H. Saltzer, D. P. Reed, and D. D. Clark. End-to-end argmuments in system design. *ACM Transactions on Computer Systems*, 2(4):277–288, November 1984.
27. Chris Sander, Reinhard Schneider, and Pieter Stouten. The human genome and high performance computing in molecular biology. In H. W. Meuer, editor, *Supercomputer'92, Anwendungen, Architekturen, Trends*, pages 32–48. Springer-Verlag, 1992.
28. Volker Strumpen. Coupling Hundreds of Workstations for Parallel Molecular Sequence Analysis. *Software — Practice and Experience*, 25(3):291–304, March 1995.
29. Volker Strumpen. *The Network Machine*. PhD thesis, Department of Computer Science, Swiss Federal Institute of Technoloy (ETH) Zürich, 1995. (URL `ftp://ftp.inf.ethz.ch/pub/publications/dissertations/th11227.ps.Z`).
30. Volker Strumpen and Thomas L. Casavant. Implementing Communication Latency Hiding in High-Latency Computer Networks. In *High-Performance Computing and Networking*, LNCS 919, pages 86–93. Springer-Verlag, Milano, Italy, May 1995.
31. Moti N. Thadani and Yousef A. Khalidi. An efficient zero-copy I/O framework for UNIX. Technical Report SMLI TR-95-39, Sun Microsystems Laboratories, May 1995.
32. C. A. Thekkath, H. M. Levy, and E. D. Lazowska. Efficient support for multicomputing on ATM networks. Technical Report 93-04-03, Department of Computer Science and Engineering, University of Washington, April 1993.
33. Robert Wahbe, Steven Lucco, Thomas E. Anderson, and Susan L. Graham. Efficient software-based fault isolation. In *14th ACM Symposium on Operating Systems Principles*, pages 203–216, Asheville, North Carolina, December 1993.
34. Alan Wood. Predicting client/server availability. *IEEE Computer*, 28(4):41–48, April 1995.

This article was processed using the LaTeX macro package with LLNCS style