A Case for NOW (Networks of Workstations)

Networks of workstations are poised to become the primary computing infrastructure for science and engineering. NOWs may dramatically improve virtual memory and file system performance; achieve cheap, highly available, and scalable file storage; and provide multiple CPUs for parallel computing. Hurdles that remain include efficient communication hardware and software, global coordination of multiple workstation operating systems, and enterprise-scale network file systems. Our 100-node NOW prototype aims to demonstrate practical solutions to these challenges.

Thomas E. Anderson

David E. Culler

David A. Patterson

and the NOW team

University of California at Berkeley

A fundamental concept in biology is the stable food chain: Big fish eat smaller fish, which in turn feed on still smaller fish. Each type of fish is adapted to its own ecological niche. Computer systems also occupy ecological niches of a sort. Personal computers and workstations are small systems, designed to provide fast and predictable interactive performance on jobs of modest size. Servers and mainframes are more expensive, oriented to more demanding applications and larger numbers of users. Supercomputers aim to achieve the ultimate in performance at any cost.

The computing food chain seems to operate in reverse. The smallest fish, personal computers, are eating the market for workstations, which in turn have consumed the market for minicomputers and are eating away at the market for larger mainframes and supercomputers. Why is this? One reason is the effect of volume manufacturing on computer price-to-performance ratios.

The rapid improvement each year in computer system performance does not happen by accident; it requires a huge investment in engineering and manufacturing. For personal computers and workstations, manufacturers can amortize this investment over a large sales volume. With much smaller sales volumes, mainframes and supercomputers must either forgo performance advances or obtain them at higher per-unit cost. Workstation price/performance ratios are improving at 80 percent per year, while those of supercomputers are improving at only 20 to 30 percent. Given that desktop computers offer the best price/performance tradeoff in this era of sustained rapid change, why would anyone buy a supercomputer? One reason is that there may be no choice: The task at hand may be bigger than will feasibly run on a workstation.

How can we exploit this transformation of the technology base toward small computers? We argue that the ongoing technological convergence of local area networks and massively parallel processor interconnections will allow NOWs to replace the entire computer food chain. (We use the term workstation to refer generically to the computer system designed for the desktop. High-end personal computers have acquired all the capabilities that once distinguished workstations, which include local area networking and a full-function operating system.)

Instead of small computers for interactive use and larger computers for demanding sequential and parallel applications, we propose using NOWs for all the needs of computer users. In particular, the Berkeley NOW project tries to harness all the computers in a building to satisfy the needs of both desktop computing and applications that require a hundredfold more computing resources than any single machine within that building can provide.

0740-7475/95/$04.00 © 1995 IEEE

Technology trends

For most of the VLSI generation, a handful of dominant technological forces have shaped computer systems design. Microprocessor performance has improved by 50 to 100 percent per year. DRAM memory and disk capacities have quadrupled roughly every three years.[1] These trends provide the basis for many abstract cost metrics and analyses of what is practical at various points in time. However, if we abstract too far away from the industry that produces the technology, we run the risk of losing sight of two critical constraints: volume and dollars.

Cheaper computers are attractive to a larger market. Personal computers are manufactured in much larger volumes than workstations or servers, which in turn are manufactured in much larger volumes than mainframes or supercomputers. Larger volume means that manufacturers can amortize the massive development costs required to sustain the rate of technological innovation over a larger number of units. Other economies of scale further contribute to the improved cost/performance ratio.

Gordon Bell summarized these effects with a rule of thumb: Doubling the volume reduces the unit cost to 90 percent. For example, over the past five years the volume of personal computers shipped per supercomputer has been about 30,000:1. Thus, Bell's rule predicts a fivefold cost advantage for the smaller system. Looking at one comparable component of these systems, we see that in January 1994 the cost per megabyte of DRAM was $40 for a personal computer and $600 for the Cray M90 family, a price multiplier of 15. The bottom line is that smaller computers offer a better cost/performance ratio than larger ones.

The interesting question is what do these cost/performance trends mean if we need more processor cycles, memory, or disk than a small system can reasonably provide? Must we buy a single computer big enough for the biggest task we ever need to run and pay a huge premium for the additional capacity? Indeed, there is a market for servers, mainframes, and supercomputers, even though they offer worse cost/performance ratios than workstations or personal computers. Most engineering workstations have a huge amount of memory and very fast processors, both of which sit idle most of the time. While building large computing systems out of small, mass-produced computers is clearly attractive, we must also make sure that we can deliver to a single task far more resources than fit in one box.

Lessons from massively parallel processors. Analyses similar to those just described led many in the mid 1980s to speculate that the "killer micro" would take over high-performance computing.[2] Today, supercomputing is led by MPPs—machines constructed as a large collection of workstation-class nodes connected by a dedicated, low-latency network. It would seem that these do exploit the commodities of "killer" technologies: a fast microprocessor, its sophisticated cache, and large, inexpensive DRAM. What has limited their success? Examining the strengths and weaknesses of MPPs will help us understand the key constraints under which NOW must achieve its goals.

Table 1. Comparison of MPPs and workstations with the same or comparable microprocessor.

MPP	Node processor	MPP year	Year of equivalent processor
T3D	150-MHz Alpha	1993-94	1992-93
Paragon	50-MHz i860	1992-93	1991
CM-5	32-MHz SS-2	1991-92	1989-90

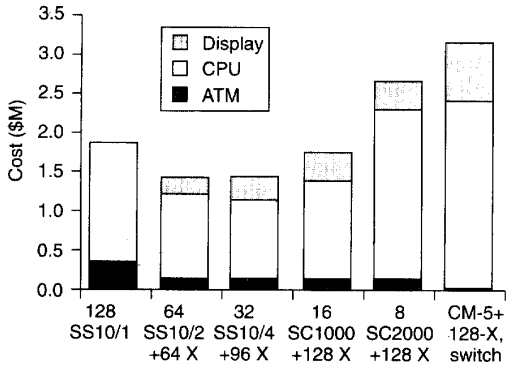

Figure 1. January 1994 price comparison (at discount) for a range of equivalent 128-processor systems.

One key weakness with MPPs is engineering lag time. With the performance of commodity components increasing rapidly, any delay between freezing the design and shipping the system subtracts from performance. As Table 1 indicates, MPP systems tend to lag one to two years behind workstations built from comparable parts. At 50-percent performance improvement per year, a two-year lag costs more than a factor of two in the bottom-line computational performance.

The increased engineering effort of a highly integrated system exacerbates the cost/performance disadvantage of low-volume manufacturing. This is not unique to MPP systems; it applies to multiprocessor servers as well. For example, Figure 1 shows the price charged to universities for a range of systems, all providing 128 40-MHz SuperSparc processors, 128×32 Mbytes of memory, 128 Gbytes of disk, 128 screens (X-terminals for multiprocessor configurations), and a scalable interconnection. The first three systems are Sparcstation-10s with one, two, or four processors. Next are SparcCenter-1000

and SparcCenter-2000 servers that can contain up to 8 and 20 processors. Last comes a 128-node MPP, either the Thinking Machines CM-5 or the Meiko CS-2. The latter systems include a large engineering effort, beyond that of the commodity parts, which a relatively small volume of sales must bear. The price is twice as high for either the large multiprocessor servers or MPPs compared to the most cost-effective workstation.

These figures indicate the trade-offs in multiprocessor system integration. Repackaging the chips on the desktop motherboard improves system density and potentially reduces parts costs. It may provide access to internal buses for the network connection, which tends to allow for better communications performance than standard peripheral points. However, the integration effort extends product lag time and increases development costs. (Our experience is that it also increases the per-node maintenance cost.) By increasing lag time, it degrades computational performance. Since the final performance of any task depends on both the communication and computation rates, the advantages of integration clearly have limits.

An often unappreciated weakness is the high cost MPPs incur in changing the operating system and other commodity software. Workstation vendors invest as much in operating-system development as they do in microprocessor design, plus a vast body of applications depends directly on the operating-system interface. Early MPPs had to provide custom message-passing kernels, and applications had to be modified for each new machine. More recent machines offer a full Unix on each node; however, repackaging the chips and eliminating typical devices (local disk, serial port, and Ethernet) has forced a split from the commodity operating-system development path. This divergence results in less functionality, lower reliability, and further increases lag time.

The final weakness is that the niche occupied by MPPs is too narrow. They successfully delivered very high performance in certain applications domains, where rewriting the application was tractable. They have not, however, provided a versatile tool delivering high throughput on general-purpose tasks, such as file service, nor have they provided fast and predictable interactive performance.

Nevertheless, as a collection of workstation-class computers, MPPs provide two main advances we need in a NOW: communication performance and a global system view. Current MPP systems provide a dedicated, high-bandwidth network that scales with the number of processors. In discussing communication performance, we must distinguish the time spent in the actual network hardware, called latency, from time spent in the processor preparing to send or receive a message, called overhead.[3] Network latency can potentially overlap with computation, while overhead is CPU time that is unavailable for computation.

MPP communication performance derives from several factors. The routing components are fast, single-chip switches employing cut-through routing with short wire delays and wide links. The network interface is close to the processor, typically on the processor-memory bus, rather than on a standard I/O bus. Even with this high-quality communication hardware, the overhead for conventional message passing on MPP systems is typically a few thousand processor cycles.[4] Although this is an order of magnitude better than typical LAN overheads, it is still substantial. Using lean communication layers, especially Active Messages,[4] reduces this software overhead by an order of magnitude. For example, on the CM-5, which provides user-level network access, the processor overhead for sending and handling a small message is about 50 processor cycles each (25 instructions, 1.7 μs). The network latency is less than 130 cycles (4 μs) across a 1,024-processor machine.

The global system view means that a single parallel program runs on a large collection of nodes as a single entity, rather than as an arbitrary collection of processes. Job control pertains to the entire collection. Files are uniformly accessible across all the nodes. Most importantly, the processes are scheduled as a single unit, so the constituents of a parallel program actually run in parallel.

Although the networking advances in MPPs represent a key breakthrough, the MPP experience shows that it is not enough to exploit commodity components. One needs to exploit the full desktop building block, including the operating system and applications. 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. These demands require a resource allocation policy that explicitly preserves interactive performance, while allowing use of dedicated and unused resources throughout the network by demanding applications: DRAM for memory-intensive programs, disks for I/O bound programs, and CPUs for CPU-bound programs.

Why NOW now? The idea of using idle resources over a network has been around nearly since computers became networked; parallel computing on clusters of workstations is hardly new.[5,6] What is new is the convergence of technologies and systems concepts that together make NOWs more attractive than ever before.

- *The "killer" network.* Switched LANs allow bandwidth to scale with the number of processors, while low-overhead communication protocols have enabled very fast communication over a LAN. These technologies have worked very effectively in MPP systems that span several tens of meters. They are emerging in the LAN arena, with asynchronous transfer mode and other recent alternatives, including the Myrinet presented in this issue of *IEEE Micro*.[7]
- *The "killer" workstation.* Workstations have become

extraordinarily powerful. A top-end 1994 workstation provides roughly one third the performance of a Cray C90 processor and exceeds C90 capacity in many respects. In addition to processor performance, a typical workstation offers large memory and disk capacity. Therefore, the resources on a desktop are more worthwhile to recruit than ever before. The key to doing so is the network hardware and software.

- *The I/O bottleneck.* Processors are getting much faster, but disks are improving mostly in capacity, not performance. If current trends continue, further increases in processor performance will yield little improvement for the end user, since more and more time will be lost waiting for I/O. NOWs offer a better alternative. A huge pool of memory potentially exists on the network; this memory can be accessed far more quickly than local-disk storage. Furthermore, when I/Os are required, they can be striped across multiple workstation disks, much as in a RAID (redundant array of inexpensive disks). The key enabling technology is again the network.

Clearly, NOWs offer advantages not just for parallel computing. They focus a large collection of resources on a single program: large memory, large disk, or large processing. The time has come to concentrate on building large systems out of high-volume hardware and software components, and to raise the level at which we research systems. Taking this approach will let us tailor future high-volume components as better building blocks for such large-scale systems.

Opportunities for NOW

A NOW offers a wide range of advantages when implemented on a building-wide scale of hundreds of machines. The pool of resources in a NOW include memory, disks, and processors. In each case, we need to ask how a NOW system can be more to the end user than simply a bunch of machines on a fast network.

Memory. Fast network communication makes it attractive to use a NOW's aggregate DRAM as a giant cache for disks. In the past, this has not been practical on Ethernet because it would consume too much of the shared-media's bandwidth. Also, even on an idle Ethernet, fetching data across the network is only marginally quicker than a local-disk access. Emerging switch-based LANs provide ample bandwidth, however, and the remote memory access time is an order of magnitude faster than that of disk.

Table 2 shows a conservative estimate of the time for an 8-Kbyte access on a DEC AXP 3000/400 on both Ethernet and ATM using standard network drivers. There is even a bigger benefit with the low-overhead network hardware and software described later in the discussion of low-overhead communication and in von Eicken et al.[8] By using the idle DRAM on a NOW, we can dramatically reduce the number of disk accesses, mitigating the I/O bottleneck, and greatly improving user-visible performance. There are two applications of this idea: virtual memory and file caching.

Table 2. Time to service a file system cache miss from remote memory or disk to Ethernet and for 155-Mbps ATM.

	Ethernet		155-Mbps ATM	
	Remote memory (µs)	Remote disk (µs)	Remote memory (µs)	Remote disk (µs)
Memory copy	250	250	250	250
Net overhead	400	400	400	400
Data transfer	6,250	6,250	400	400
Disk	—	14,800	—	14,800
Total time	6,900	21,700	1,050	15,850

Network RAM. Virtual memory was introduced to run problems much bigger than main memory. The idea was to automatically migrate data between main memory and slower, cheaper storage, giving the illusion of a large, inexpensive memory. Unfortunately, as the performance gap between processor and disk widened, this illusion has broken down. Today, people arrange never to run problems bigger than the physical memory of the machine. To run a larger program, typically one needs to buy more DRAM or, if no more will fit, find a bigger (and less cost-effective) computer that can hold more DRAM. This happens despite the presence of gigabytes of idle DRAM on the network.

Network RAM can fulfill the original promise of virtual memory. With high-bandwidth, low-latency networks and system software that can recognize when machines are idle, we can page effectively across the network. Simulations, such as the one shown in Figure 2 (next page), suggest that programs run 10 to 30 percent slower using network RAM than if the program fits entirely in local DRAM. Using network RAM, however, is 5 to 10 times faster than thrashing to disk.

Cooperative file caching. Analogously, we can improve file system performance by cooperatively managing the file caches on each client workstation. Traditionally, network file systems reduce network accesses by caching files in local client memory; they reduce disk accesses by caching files in server memory. In a building-wide NOW, the aggregate client memory far outstrips the memory the server can feasibly contain.

Cooperatively managing this large client memory has two benefits. First, more than one client uses a number of files, such as executables and font files. On a cache miss, the system can fetch files from another client's memory, instead of going to the server's disk. Second, active clients can effectively increase the size of their cache by using the memory of idle clients.

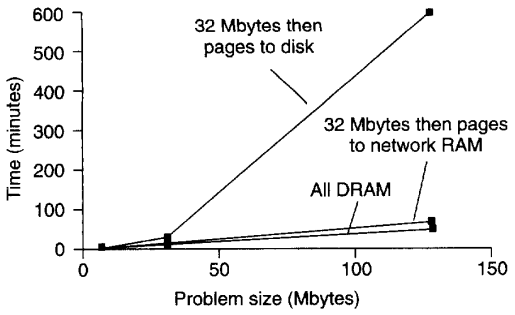

Figure 2. Estimated execution time as the size of the multigrid problem increases running on three systems: a workstation with 32 Mbytes of DRAM plus disk; one with 128 Mbytes of DRAM; and one with 32 Mbytes plus paging to DRAM on other machines on the network.

Table 3. Impact of cooperative caching: 42 workstations, 16-Mbytes/workstation, 128 Mbytes/server.

	Cache miss rate (percent)	Read response time (ms)
Client server	16	2.8
Cooperative caching	8	1.6

We have investigated the potential benefits of cooperative caching in file systems by examining a two-day trace of file system activity on a cluster of 42 workstations at Berkeley. Table 3 shows the simulated results for a practical implementation of cooperative caching, including the overhead of coordinating the contents of the various caches. Assuming each client workstation has 16 Mbytes of file cache and the server cache is 128 Mbytes, cooperative caching reduces disk reads by a factor of two, improving file read performance by 80 percent.[9]

Redundant arrays of workstation disks. RAID systems deliver higher bandwidth, capacity, and availability than a single large disk can achieve by hooking together arrays of small disks. However, RAID systems suffer from some drawbacks. The cost per byte of disk storage is often worse than single disks by a factor of two, due to the hardware needed to manage the RAID. Further, the RAID must connect to a host computer, which is often a performance and availability bottleneck. Although RAID systems use redundant storage to ensure a large mean time to failure, if the host computer crashes, the RAID becomes unavailable.

NOWs let us address these issues. Instead of building the RAID in hardware, we can build it in software, writing data redundantly across an array of disks in each of the network's workstations. Effectively, the fast network needed for network RAM and cooperative file caching can also serve as the I/O backplane of a RAID system. By striping across enough disks, each workstation can appear to have disk bandwidth limited only by the network link bandwidth. Parallel programs can achieve the aggregate disk bandwidth of the entire cluster. Availability of a software RAID on a NOW could be better than in a hardware RAID system, because there is no central host to be a single point of failure. If one workstation in the NOW crashes, any other can take its place in controlling the RAID.

Parallel computing. NOWs also let us support high-performance parallel applications within an everyday computing infrastructure. For many real-world applications, we need processors capable of high, sustained floating-point performance, networks with bandwidth that scales with the number of processors, parallel file I/O, and low-overhead communication. One example is the NASA Ames/UCLA chemical tracer model called Gator;[10] it models atmospheric chemistry in the Los Angeles basin and has served for detailed air pollution studies.

Demmel and Smith[10] have developed a model of Gator's execution time as a function of various input parameters (grid resolution, number of chemical species) and system parameters (CPU floating-point performance, number of CPUs, message bandwidth and overhead, file I/O bandwidth). Studies have validated the predicted wall-clock times for the computation portion of the application to within 30 percent against measured times on a 16-node Cray C-90, a 64-node CM-5, and a 9-node DEC Alpha workstation farm.

Table 4 shows the results of this model for several machine configurations. The transport phase is communication intensive, while the ordinary differential equation phase (ODE) is highly parallel. The computation involves 36 billion floating-point operations; the run needs 3.9 Gbytes of input; and 51 Mbytes of output result. We consider a 16-node C-90 (300-Mflops and 10-Mbytes/s disk per CPU), a 256-node Paragon (12-Mflops and a 2-Mbytes/s disk per node), and a number of hypothetical 256-node RS/6000 NOWs (40-Mflops and a 2-Mbytes/s disk per node). The baseline NOW system assumes Ethernet, PVM (Parallel Virtual Machines, a popular message-passing system for workstations), and a sequential file system. The performance of this system is dreadful, taking three orders of magnitude longer than the Paragon or C-90.

Sharing a single Ethernet among a large number of high-performance processors limits the performance. Upgrading to a higher bandwidth ATM network dramatically improves performance of the transport phase, improving overall performance by an order of magnitude. But the bandwidth of

the sequential file system (2 Mbytes/s) still limits us. Adding a parallel file system that delivers 80 percent of the aggregate bandwidth of the workstation disks improves overall performance by yet another order of magnitude. Finally, replacing PVM with a low-overhead, low-latency communication system further reduces the execution time by an order of magnitude, to where performance on the NOW competes successfully with the C-90 at a fraction of the cost. The performance is better than on the Paragon, because the floating-point performance of commercial workstations greatly exceeds that of a single node on an MPP. In summary, we need good floating-point performance, scalable network bandwidth, a parallel file system, and low-overhead communication to deliver high performance for this application.

Table 4. Predicted execution time in seconds for 12-hour Gator simulation on a Vector Supercomputer, MPP supercomputer, and four versions of NOW.

Machine	ODE (s)	Transport (s)	Input (s)	Total (s)	Cost ($M)
C-90 (16)	7	4	16	27	30
Paragon (256)	12	24	10	46	10
RS-6000 (256)	4	23,340	4,030	27,374	4
RS-6000 + ATM	4	192	2,015	2,211	5
RS-6000 + parallel file system	4	192	10	205	5
RS-6000 + low-overhead msgs	4	8	10	21	5

Workloads of a building-wide system. The measurements just discussed suggest that NOWs can work well as dedicated systems. We can also use the infrastructure supporting interactive work for demanding applications. The key question is whether a NOW can run large programs with the performance of a dedicated large computer and run small programs with the interactivity of a dedicated workstation. To investigate this combination, we simulated the impact that sequential workstation and MPP jobs may have on one another.[11]

We collected traces from a local cluster of 53 DECstation 5000/133s with 64 Mbytes of memory used by electrical engineering graduate students. Two user-level daemons logged information every two seconds on CPU, memory, disk, keyboard, and mouse activity. We collected data for two months, resulting in roughly 3,000 workstation-days of traces. We randomly selected the workstation traces used in our simulations from different weekday traces, allowing us to simulate a cluster of more than 53 workstations. For our parallel-machine trace, we obtained a month's worth of data on parallel jobs from a CM-5 at Los Alamos National Laboratories. The trace consists of a mix of production, and development runs on a 32-node system.

We found that even during the daytime hours, more than 60 percent of workstations were available 100 percent of the time. (We call a machine *available* if there is no user activity or active jobs for one minute.) This finding runs directly counter to the popular belief that idle machines are only available during off hours. Since idle workstations are available, the question then is how many workstations does someone need to run the MPP workload without interfering with the workstation users? Figure 3 shows that for these traces, assuming equal network and CPU performance in the NOW and the MPP, the parallel workload of a 32-node MPP runs only 10 percent slower when running on 64 workstations that are handling a typical sequential workload as well. This is like getting almost a CM-5 for free.

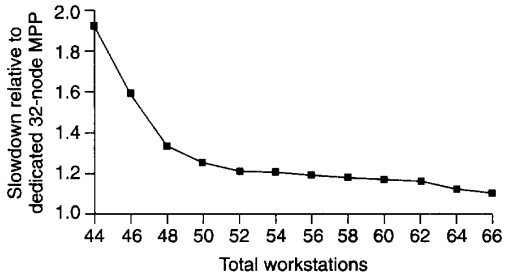

Figure 3. Slowdown of 32-node MPP workload from LANL running on a NOW running a sequential workload as the number of workstations in NOW increases.

Summary of opportunities. A NOW system offers more than a collection of workstations on a fast network. It affords the opportunity to make advances in traditional system functions, such as virtual memory and file systems, as well as parallel computing. In obtaining higher performance from a NOW, our approach may differ from that typically found with MPPs.

- In our approach, we would avoid going to disk by using all the DRAM on the network.
- If the application were still not fast enough, we would try using all the disks on the network to speed up the remaining I/O.
- If it were still not fast enough, we would parallelize the computational portion.

This layered approach seems more attractive than the traditional first step required of MPP users: completely rewriting the program before seeing any benefit from the machine.

The Berkeley NOW project

Recognizing the rapid advance and tremendous investment in such technologies, the Berkeley NOW project has used commercial, off-the-shelf systems wherever possible in realizing the opportunities of NOW. As a research vehicle, following this principle affords two additional advantages: The implementation is quicker if we avoid reinvention, and exploiting mostly off-the-shelf technology will simplify the transfer of new ideas and technology.

Low-overhead communication. Bandwidth is the widely advertised metric of communication performance. However, network latency and processor overhead can be just as important, despite their low profile in product literature. This is a peculiar oversight because processor overhead is the dominant factor determining the communications performance of real programs.

Several studies have examined the communications characteristics of parallel programs. As in the earlier Gator example, many important programs transfer many small messages and are sensitive to communications overhead. What is perhaps more surprising is that many conventional LAN applications exhibit similar characteristics.

We obtained a trace of network file system traffic over one week from 230 clients of our departmental file servers. Although file transfers are performed in large blocks, we found that 95 percent of the NFS messages are less than 200 bytes, due to queries to the file system metadata. Moreover, these queries must complete before file data can be transferred, so NFS performance is directly coupled to the round-trip message time—the overhead and latency.

On Sun Sparcstation-10s connected by Ethernet, we measured 456 µs of processor overhead plus (unloaded) network latency on a single message and a peak bandwidth of 9 Mbps through TCP/IP. With the same processors and a Synoptics ATM network, the bandwidth increases to 78 Mbps, but the overhead plus latency also *increases* to 626 µs. (Note that the Synoptics performance is within 20 percent of most other ATM networks. The network latency component varies for different switches from about 10 to 100 µs, depending on the specific configuration. The network interface adapter adds as much as 100 µs to the latency, but the largest fraction of the time is the processor overhead resulting from the system software.)

If we apply these coefficients to our trace, the eightfold increase in bandwidth reduces the data transmission time component dramatically but the overall improvement is just 20 percent because the overhead plus latency component remains large. This example illustrates that emerging high-bandwidth network technologies will provide a major advance only if they are accompanied by corresponding reductions in latency and processor overhead.

Our target is to perform user-to-user communication of a small message among one hundred processors in 10 µs. This goal is technologically feasible, but leaves very little room for compromise. For example, that target figure equals the processor overhead plus network latency on the current CM-5, plus a single serialization delay of an ATM cell. Several aspects of a NOW make us optimistic about meeting this goal, while others make it quite challenging. In NOW we will have faster processors, but greater constraints on where the network connects to the node. We will have somewhat higher link bandwidth, but may have greater routing delay and uncertain reliability. The nodes support a full Unix system, with relatively rigid device and scheduling interfaces.

Our work focuses on the network interface hardware and the interface into the operating system. To meet our goal, the user must transmit directly into and receive from the network, without operating-system intervention. Therefore, the operating system must map data and control access to the network interface into the user address space. The network interface must establish the communication protection domain, which it can do by inserting a network process ID into each outgoing message and checking each incoming message. It also needs the ability to deliver data and notification directly into the user process, at least for the currently running process. If the message is going to awaken a process, other aspects of the notification process will dominate. We do need to buffer messages properly in the meantime, however. Furthermore, the network interface hardware will likely need to assist in supporting message loss as an infrequent case.

One initial prototype is a cluster of HP9000/735s using an experimental Medusa FDDI network interface that connects to the graphics bus and provides substantial storage in the network interface. As Martin describes,[12] using user-level Active Messages provides a processor overhead of 8 µs, including time-out and retry support. This overhead figure also includes almost 3 µs of processing that is entirely an FDDI artifact. The network and adapter latency adds an additional 8 µs.

We obtain the full link bandwidth for large transfers and half of the peak bandwidth on 175-byte messages, compared to 760-byte messages for single-copy TCP and 1,350 for standard TCP. Constructing conventional sockets on top of this layer, we see a one-way message time of about 25 µs, nearly an order of magnitude faster than TCP or single-copy TCP on the same hardware.

Our final demonstration system will use either a second-generation ATM LAN or a retargeted MPP network, such as the Myrinet.[7] We are currently evaluating a spectrum of design alternatives for the network interface card, which will connect either at an emerging high-speed external bus, such as PCI, the memory bus, or the graphics bus, depending on our final choice of workstation platform.

The key differences in this work, compared to traditional LAN interfaces, are the low-overhead, low-latency communication orientation; the quality of the interconnection itself and

the simplicity that derives from that; and the recognition that we have some control over all the nodes that attach to the network and thus can make strong assertions about the endpoints.

GLUnix: A global layer Unix. The second key challenge is effective management of the pool of resources within a NOW. The idea of globally managing network resources has been around for a long time, yet the most widely used commercial systems do not provide this service. This lack of progress results in part from two significant impediments: implementing global services in the context of existing commercial operating systems and the sociology of global resource sharing.

GLUnix structure. Recall that the hardware argument for NOWs is that we should build large-scale computer systems by networking together small, yet complete, mass-produced commercial systems. The same is true for software. There is a tremendous advantage to leveraging the hundreds of millions of dollars invested each year in commercial operating-system development, not to mention the billions invested in application development for these systems.

Nevertheless, the typical first step for operating-systems research projects is to throw out the commercial system and start from scratch. This is often conceptually easier because it avoids cumbersome artifacts of a working body of code, but building a real working system means reimplementing a huge amount of incidental code (device drivers, virtual memory management, process dispatching, and so on) that already works in commercial systems.

Instead, our approach provides the global services of a NOW by gluing together local Unixes running on each workstation on the network. As much as possible, our approach builds this GLUnix as a layer on top of unmodified commercial Unixes. (Although we are implementing GLUnix as a layer on top of Unix, nothing in our approach depends on Unix as a building block. We could as easily build GLUnix as a layer on top of personal computer operating systems, such as Windows NT or even DOS.) By leveraging the complete workstation, including the local operating system, we had a working prototype of GLUnix after only a three-month effort. This layered approach also better lends itself to tracking advances in the underlying commercial system. The challenge, of course, is performance.

A key technology that allows us to layer efficiently on top of existing systems is software fault isolation.[13] Traditionally, operating-system kernels (other than on personal computers) use hardware virtual memory to enforce firewalls between user applications. Recent work demonstrates that we can efficiently implement the same firewalls in software, by modifying the application object code to insert a check before every store and indirect branch instruction. By applying aggressive compiler optimization techniques, the overhead of enforcing firewalls in software can fall to between 3 and 7 percent on several of today's RISC processors. For the same overhead, we can insert a protected virtual operating-system layer into any Unix application entirely at the user level. This layer catches and translates the applications system calls, to provide the illusion of a global operating system. For example, we use software fault isolation to implement completely transparent process migration and global resource scheduling.

Of course, providing all the global services a user might want with absolutely no kernel changes is impossible. Our goal is to look for the minimal set of changes necessary to make existing commercial systems NOW ready. One example is replacing the kernel communication software with a low-overhead implementation. Another is using network RAM within the virtual-memory system; we can implement this most easily by replacing the swap device driver, an operation supported at the user level by some, but not all, modern Unix systems. As long as the required changes are small, it is feasible to get them included in commercial systems. For example, despite the lack of a compelling market for parallel programs, several years ago industry added synchronization operations (such as test&set) to processor instruction sets to make their hardware "parallel ready."

GLUnix sociology. Perhaps the largest roadblock to the success of NOW is the sociology of sharing computing resources. Interactive users look suspiciously at NOW, fearing that demanding applications will steal resources and hurt their interactive response time. After all, one of the principal benefits of the move from timesharing to desktop computers a decade ago was the guarantee of a computer to each user. At the same time, supercomputer users also look suspiciously at NOW, fearing that interactive users will have priority and demanding applications will only be allowed to run at night. Like anyone else, supercomputer users work during the daytime and therefore need good response time even during the daytime.[11] GLUnix needs to address both of these concerns, along with being tolerant of individual node failures. We discuss each of these issues in turn.

We guarantee at least the performance of a stand-alone workstation to every active user, by migrating external processes off an idle machine when the user returns.[14] The key to making this approach practical is to consider not only CPU cycles, but memory contents as an interactive resource. On current Unix systems, if a demanding application runs on an idle workstation, it will eventually flush out its virtual-memory pages and file cache contents. When the user returns, the workstation's response time will visibly slow as its working set pages back in from disk. (We know of one site where a system was in place to use idle machines for distributed compiles, and people would tap their keyboards periodically simply to keep their memory contents from disappearing!)

Instead, we intend to explicitly save the idle machine's memory contents before using it, so that we can return the machine to the exact state it was in before it went idle. This

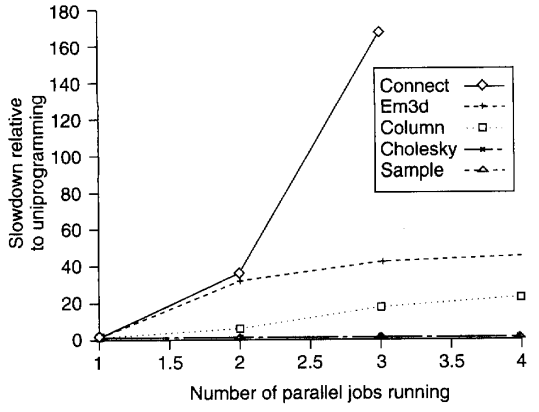

Figure 4. Impact of local scheduling on parallel-program performance referenced to coscheduling.

is feasible because of the combination of technologies in NOW. With ATM bandwidth and a parallel file system, 64 Mbytes of DRAM can be restored in under 4 seconds. To further reduce complaints by interactive users, we explicitly limit the number of times per day external processes can delay any interactive user.

We also need to deliver a large portion of the system's aggregate capacity to demanding applications. One issue is that MPP operating systems are typically specialized for scheduling parallel applications, whereas NOWs have independent Unix kernels on each processor. This local scheduling, employed by parallel environments such as PVM, requires no system support. However, it leads to unacceptable performance for processes that communicate frequently.

For example, Figure 4 shows the slowdown with local scheduling, compared to coscheduling all processes of a parallel application,[15] as the number of competing parallel jobs increases. Two of the applications send many small messages to random processors, but as long as enough buffering exists on the destination processor, the sending processor is not significantly slowed. The Column benchmark test runs slowly even though it communicates infrequently, because it overflows the buffers on the destination. Em3d suffers from delays encountered at synchronization points and Connect performs very poorly because processors frequently require data from other processors.

Because many parallel programs run as slowly as their slowest process, a machine running the parallel job while simultaneously serving for interactive computing can also compromise parallel performance. Thus, we need to migrate demanding jobs off no longer idle machines to preserve both interactive and parallel performance. Fortunately, our measurements, along with those of others, indicate that a large fraction of workstations are idle, even at the busiest times of the day. Consequently, there will usually be a machine to which the evicted process can migrate.

The ability to quickly move processes between machines, along with their memory state, is important for parallel-program performance as well as reducing delay for the interactive user. While one process is migrating, the rest of the parallel program is unlikely to make much progress. The study we discussed earlier indicates that by implementing fast process migration and choosing idle machines that are likely to stay idle, we can overlay a typical 32-processor parallel workload on a 64-workstation cluster used for interactive jobs without significantly sacrificing the performance of either. An organization with a more demanding workload would simply have to extend the capacity of its NOW with additional noninteractive machines.

A final consideration in GLUnix is that the system must continue to operate in the face of individual node crashes, new resources being added or deleted from the network, and even operating-system software upgrades. On today's multiprocessors, if any CPU fails (or its operating-system software crashes), users must reboot the entire system. Similarly, they must take the entire multiprocessor out of service to upgrade its hardware or software.

This situation makes it impractical to use an MPP or large server as the sole computing infrastructure for a building, since any small thing going wrong would inconvenience all users. If a workstation fails in our model, it only affects the programs using that CPU; those programs can restart from their last checkpoint, while programs running on other CPUs continue unaffected. We are also structuring our software to tolerate hot-swap upgrades of hardware and software.

Many consider security to be the Achilles' heel of NOWs. If malicious users can compromise the local operating system on any machine in the NOW, they can corrupt any process or data migrated to that machine. However, many organizations enforce physical security at the level of the entire building, rather than the individual machine. We assume that resource sharing within a NOW will only be used within a single administrative security domain. In addition, a small amount of hardware in the network interface can ensure that the correct operating system is booted on a machine, before allowing it to connect into the NOW. When tighter security is required, users can always remove the collection of machines from the desktop and place it in the machine room, with X-terminals on the desktop. But the same basic problems remain no matter where the workstations are physically located. Unlike the mainframes of the past, we must guarantee as good performance as a standalone workstation to interactive users by retaining their cached state, and we must provide effective scheduling of parallel applications.

xFS: Serverless network file service. Client-server computing has become a popular way of structuring distributed systems. In most network file systems, a central server machine provides the abstraction of a single file system shared among the users logged into a number of client workstations. Disks at the server store files; clients can access them via requests made over the network.

Unfortunately, a central server design has performance, availability, and cost drawbacks. Any centralized resource will become a bottleneck with enough users. In traditional network file systems, even if clients cache frequently used files, all cache misses and all modified data go to the server, ultimately limiting scalability. Furthermore, any client DRAM and disk storage only benefits a single user. Partitioning the file system among multiple servers will let the system support more users, but this requires the system manager to effectively become *part* of the file system, moving users, volumes, and disks between servers to balance the load. Similarly, a central server is a single point of failure, requiring the expense of replicating the server to provide good availability. Perhaps most importantly, as Figure 1 showed, server machines are expensive: Memory and disks are cheaper in a workstation, even ignoring the cost of server replication for high availability.

In the NOW project, we are addressing these problems by building a completely serverless network file system, called xFS. In place of a centralized server (or set of replicated servers), client workstations cooperate in all aspects of the file system storing data, managing metadata, and enforcing protection. The xFS goal is high-performance, highly available network file service that can scale to an entire enterprise, at low cost.

To achieve this goal, xFS combines four features not found in other file systems:

- Any piece of the file system data, metadata, and control can dynamically migrate between clients and between storage levels; this vastly simplifies both load balancing and failure recovery (any client can take over for any failed client).
- We use shared-memory multiprocessor-style cache coherence, specifically a write-back ownership protocol, to maximize locality of control and data.
- We store file data and metadata in a software RAID, a much simpler and cheaper approach to high availability than server replication, at the same time delivering high-bandwidth disk I/O to sequential and parallel applications (see the discussion of redundant arrays of workstation disks).
- We cooperatively manage client caches as a giant cache for disk and client disk as a giant cache for robotic tape storage, to reduce the I/O bottleneck (see the discussion of cooperative file caching).

THE DRAMATIC RATE OF ADVANCE in small desktop systems dominates computer system design today, because only these systems offer the large volume and efficiency of production to support a massive, ongoing investment in architectural innovation. Thus, large-scale systems must exploit the desktop system hardware and software as a building block, rather than compete with it. The key enabling technology for this higher order style of design is a scalable, high-bandwidth, low-latency network and a low-overhead network interface. Coupled with a global operating system layer, the speed of the network allows us to view the vast collection of resources on the network processors, memories, and disks as a shared pool. This view opens up new approaches to traditional system services, including virtual memory, file caching, and disk striping, as well opportunities for large-scale parallel computing within an everyday computing infrastructure.

The challenge is to provide the individual user with the fast and predictable response time of a dedicated workstation, while allowing tasks that are too large for the desktop to recruit resources throughout the network. The raw performance of the network provides part of the solution, but we must pay careful attention to memory as an interactive resource and to scheduling assumptions in parallel programs. Examination of typical usage characteristics of dedicated workstations and of dedicated MPPs indicates that the two kinds of workloads can combine in complementary ways, given the ability to detect idle resources and to migrate processes judiciously and quickly. The latter depends critically on a fast network and a parallel file system built out of the workstation disks on that network. By exploiting this confluence of technological advances, we believe NOWs will be the systems of choice for large-scale computing within a decade.

Acknowledgments

The NOW team includes Remzi Arpaci, Satoshi Asami, Tony Chan, Mike Dahlin, Andrea Dusseau, Doug Ghormley, Seth Goldstein, Kim Keeton, Lok Liu, Steve Lumetta, Ken Lutz, Cedric Krumbein, Alan Mainwaring, Rich Martin, Jeanna Neefe, Steve Rodrigues, Drew Roselli, Amin Vahdat, Keith Vetter, Randy Wang, Kristin Wright, and Chad Yoshikawa. We are indebted to Terry Lessard-Smith, Bob Miller, and Eric Fraser for terrific administrative and technical support.

The US Advanced Research Projects Agency (F-30602-95-C-0014), the National Science Foundation (CDA-9401156), and the California Micro program have supported the NOW project. NSF Presidential Faculty Fellowships support Tom Anderson and David Culler. Hewlett-Packard, IBM, Sun Microcomputer, Digital Equipment Corp., Intel, Thinking Machines, Synoptics, Cisco, Xerox, AT&T, Siemens, Fujitsu, and Exabyte have also provided valued support.

References

1. D.A. Patterson and J.L. Hennessy, *Computer Architecture: A Quantitative Approach*, Morgan Kaufmann, San Mateo, Calif., 1990.
2. E.D. Brooks III, "Massive Parallelism Overcomes Shared-Memory Limitations," *Computers in Physics*, No. 2, Mar. 1992, pp. 139-145.
3. D.E. Culler et al., "LogP: Towards a Realistic Model of Parallel Computation," *Proc. Fourth ACM SIGPLAN Symp. Principles and Practice of Parallel Programming*, Assn. for Computing Machinery, New York, 1993.
4. T. von Eicken et al., "Active Messages: A Mechanism for Integrated Communication and Computation," *Proc. 19th Ann. Int'l Symp. Computer Architecture*, IEEE Computer Society Press, Los Alamitos, Calif., May 1992, pp. 256-267.
5. M. Mutka and M. Livny, "The Available Capacity of a Privately Owned Workstation Environment," *Performance Evaluation*, Vol. 12, No. 4, July 1991, pp. 269-284.
6. S. Zhou et al., "Utopia: A Load Sharing Facility for Large, Heterogeneous Distributed Computing Systems," Tech. Report CSRI-257, University of Toronto, Ontario, 1992.
7. N. Boden et al., "Myrinet A Gigabit per Second LAN," *IEEE Micro*, this issue, pp. 29-36.
8. T. von Eicken, A. Basu, and V. Buch, "Low-Latency Communication over ATM Networks Using Active Messages," *IEEE Micro*, this issue, pp. 46-54.
9. M. Dahlin et al., "Cooperative Caching: Using Remote Client Memory to Improve File System Performance," *Proc. First USENIX Symp. Operating Systems Design and Implementation*, 1994, pp. 267-280.
10. J. Demmel and S. Smith, "Parallelizing a Global Atmospheric Chemical Tracer Model," *Symp. High Performance Computing and Communications*, 1994.
11. R. Arapaci et al., "The Interaction of Parallel and Sequential Workloads on a Network of Workstations," Tech. Report, UC Berkeley Computer Science Dept., 1994.
12. R. Martin, "HPAM: An Active Message Layer for a Network of HP Workstations," Hot Interconnects II, Aug. 1994, available from authors.
13. R. Wahbe et al., "Efficient Software-Based Fault Isolation," *Proc. 14th ACM Symp. Operating System Principles*, ACM, 1993, pp. 203-216.
14. M. Theimer, K. Landtz, and D. Cheriton, "Preemptable Remote Execution Facilities for the V System," *Proc. 10th ACM Symp. Operating System Principles*, ACM, 1985, pp. 2-12.
15. J. Ousterhout, "Scheduling Techniques for Concurrent Systems," *Proc. Third Int'l Conf. Distributed Computing Systems*, CS Press, 1982, pp. 22-30.

Thomas E. Anderson is an assistant professor in the Computer Science Division at the University of California, Berkeley. His interests include operating systems, computer architecture, high-speed networks, massive storage systems, and computer science education. He received a PhD from the University of Washington. In 1994, he won the NSF Presidential Faculty Fellowship and the Alfred P. Sloan Research Fellowship. He has coauthored award papers at the Sigmetrics Conference, the Symposium on Operating Systems Principles, the Conference on Architectural Support for Programming Languages and Operating Systems, and the Winter and Summer Usenix Conferences. He is a member of the IEEE Computer Society and the ACM.

David E. Culler teaches computer architecture and parallel processing at the University of California, Berkeley. His research addresses parallel computer architecture, parallel programming languages, and high-performance communication structures. He has worked extensively on resource management in dataflow systems, compilation of lenient parallel languages using a Threaded Abstract Machine, fast communication on modern parallel machines using Active Messages, and Split-C. Culler received his PhD from MIT. He was a NSF Presidential Young Investigator and received the Presidential Faculty Fellowship. He is a member of the IEEE, ACM, and Sigma Xi. He was program cochair for Hot Interconnects II and is a guest editor of this issue of *Micro*.

David A. Patterson holds the E.H. and M.E. Pardee Chair of Computer Science at the University of California, Berkeley. He currently chairs both the Computing Research Association and the ACM Special Interest Group in Computer Architecture. He received the ACM Outstanding Educator Award and the University of California Distinguished Teaching Award. At Berkeley, he led the design and implementation of RISC I, which became the Sparc architecture, and also led the RAID project. His current research interests are in large-scale computing using NOWs. Patterson received a PhD in computer science from the University of California, Los Angeles. He is a member of the National Academy of Engineering and is a Fellow of both the IEEE and the ACM.

Direct questions concerning this article to David A. Patterson, Computer Sciences Division, University of California, Berkeley, CA 94720; patterson@cs.berkeley.edu; http://now.cs.berkeley.edu.

Reader Interest Survey

Indicate your interest in this article by circling the appropriate number on the Reader Service Card.

Low 168	Medium 169	High 170