A Case for NOW (Networks of Workstations)

Thomas E. Anderson, David E. Culler, David A. Patterson, and the NOW team

Abstract: In this paper, we argue that because of recent technology advances, networks of workstations (NOWs) are poised to become the primary computing infrastructure for science and engineering, from low end interactive computing to demanding sequential and parallel applications. We identify three opportunities for NOWs that will benefit end-users: dramatically improving virtual memory and file system performance by using the aggregate DRAM of a NOW as a giant cache for disk; achieving cheap, highly available, and scalable file storage by using redundant arrays of workstation disks, using the LAN as the I/O backplane; and finally, multiple CPUs for parallel computing. We describe the technical challenges in exploiting these opportunities – namely, efficient communication hardware and software, global coordination of multiple workstation operating systems, and enterprise-scale network file systems. We are currently building a 100-node NOW prototype to demonstrate that practical solutions exist to these technical challenges.

Keywords: Networks of Workstations, Communications, Parallel Computing, Message Passing, File Systems, Network Virtual Memory, Global Resource Management, Availability

1. Introduction

A fundamental concept in biology is the stable food chain: big fish eat smaller fish, which in turn feed on still smaller fish, and so on. 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. Finally, supercomputers are designed to achieve the ultimate in performance at any cost.

The computing food chain seems to operate in reverse, however: the smallest fish, personal computers, are eating the market for workstations, which have consumed the market for minicomputers and are eating away at that for larger mainframes and supercomputers. Why is this? One reason is the effect of volume manufacturing on computer price-performance. 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, this investment can be amortized over a large sales volume. With much smaller sales volume, mainframes and supercomputers must either forgo performance advances or obtain them at higher per-unit cost. Workstation price-performance is improving at 80% per year, while that of supercomputers is improving at only 20-30% per year. Given that desktop computers offer the best price-performance in this era of sustained rapid change, why would anyone buy a supercomputer? One reason is there may be no choice: you have a task that is bigger than will feasibly run on a workstation.

How can we exploit this transformation of the technology base towards small computers? We argue that the on-going technological convergence of local area networks and massively parallel processor interconnects will allow networks of workstations (NOWs) to replace the entire computer food chain.[1] 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 need a hundredfold more computing resources than found in any single machine within that building.

In Section 2 we discuss the technological and economic factors motivating our investigation into NOW systems, paying particular attention to the lessons learned from massively parallel processing systems. In Section 3 we examine the new opportunities that are enabled by a NOW system with a fast, scalable interconnect. In Section 4 we discuss some of the key technical challenges to realizing these opportunities and how they are being addressed in the Berkeley NOW project.

2. Background

2.1 Technology Trends

For most of the VLSI generation, a handful of dominant technological forces have shaped the design of computer systems. Microprocessors have been improving in performance at a rate of 50% to 100% per year. DRAM memories and disks have been quadrupling in capacity roughly every three years [PaHe90]. These trends provide the basis for many abstract cost metrics and analyses of what is practical at various points in time. However, the danger in abstracting too far away from the industry that produces the technology is losing sight of two critical constraints: volume and dollars.

Cheaper computers are attractive to a larger market. PCs are manufactured in much larger volumes than workstations or servers, which in turn are in much larger volume than mainframes or supercomputers. Larger volume means that the massive development costs required to sustain the rate of technological innovation is amortized over a larger number of units. Other economies of scale further contribute to the improved cost-performance. Gordon Bell has attempted to summarize these effects with the rule of thumb: doubling the volume reduces the unit cost to 90%. For example, over the past five years the volume of PCs shipped per supercomputer is about 30,000:1. Thus, this rule predicts a cost advantage of a factor of 5 for the smaller system. Looking at one comparable component of these systems, we see that in January 1994 the cost per MB of DRAM memory was $40 for a PC and $600 for the Cray M90 family, a price multiplier of 15. The bottom line is that smaller computers offer better cost-performance than larger computers.

1. We use the term workstation to refer generically to the computer system designed for the desktop. High-end personal computers have acquired all of the capabilities that once distinguished workstations, which include local area networking and a full function operating system.

The interesting question is what do these cost-performance trends mean if we need more processor cycles, memory and/or disk than can be reasonably provided in a small system. 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 than workstations or PCs. Most engineering workstations have a huge amount of memory and very fast processors, both of which sit idle most of the time. It is clearly attractive to consider building large computing systems out of small mass-produced computers, but we must also ensure that we can deliver to a single task far more resources than fit in one box.

2.2 Lessons from MPPs

Analyses similar to those above led many to speculate in the mid 80s that the “killer micro” would take over high-performance computing [Brooks]. Today supercomputing is led by massively parallel processors (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 commodity “killer” technologies: a fast microprocessor, its sophisticated cache, and large inexpensive DRAM. What has limited their success? Examining the strengths and weaknesses of the MPPs will help us understand the key constraints under which NOW must achieve its goal.

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

MPP	Node Processor	MPP Year	Year of Equivalent Workstation
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

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

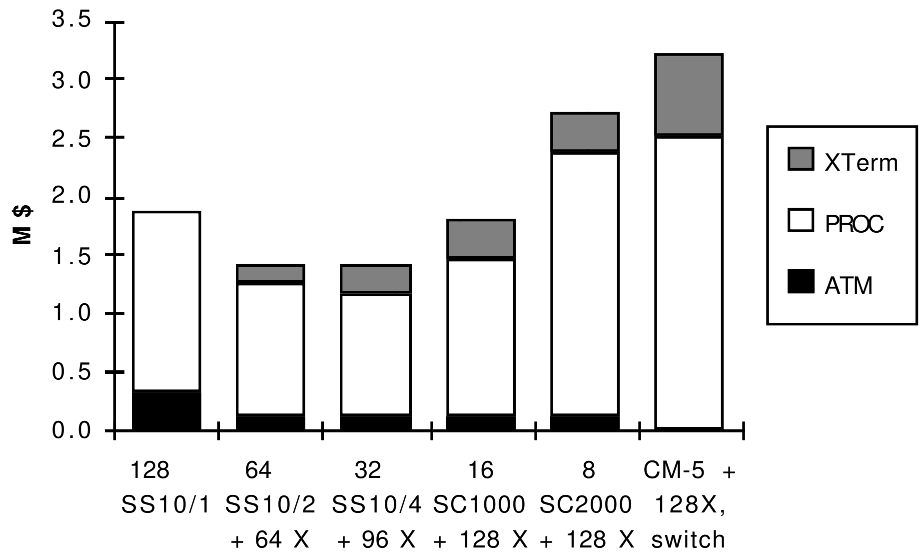

FIGURE 1. Price comparison (at discount) for a range equivalent 128 processor systems.

The increased engineering effort of a highly integrated system exacerbates the cost-performance disadvantage of low volume. This is not unique to MPP systems; it applies to multiprocessor servers as well. As an example, Figure 1 shows the university price of a range of systems all providing 128 40 MHz SuperSPARC processors, 128x32 MB of memory, 128 GB of disk, 128 screens, and a scalable interconnect. 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, respectively, and finally a 128 node MPP, either the Thinking Machines CM-5 or the Meiko CS-2. The latter systems include a large engineering effort, over and above that of the commodity parts, which must be borne by a relatively small volume of sales. The price is a factor of two higher 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 results in better system density and potentially reduces parts costs. It may provide access to internal busses for the network connection, which tends to allow for better communication performance than standard peripheral points. However, the integration effort adds to product lag time and increases development costs. (Our experience is that it also increases the per node cost of maintenance.) By increasing lag time, it hurts computational performance. Since the final performance of any task depends on both the communication and computation rates, there is clearly a limit to the advantages of integration.

A weakness that is often unappreciated is the high cost incurred by MPPs of 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 were forced 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 OS development path. This divergence results in less functionality, lower reliability, and further increased lag time.

The final weakness is the niche that the MPPs have ben able to occupy. They are successful at delivering very high performance in certain applications domains, where rewriting the application was tractable, but they have not 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, the 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 it is important to distinguish the time spent in the actual network hardware, which we call *latency*, from that spent in the processor in preparing to send or receive a message, which we call *overhead*[Cul*93]. Network latency can potentially be overlapped with computation, while overhead is processor cycles that cannot be used 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[vEi*92]. Although this is an order of magnitude better than typical LAN overheads, it is still substantial. Lean communication layers, especially Active Messages [vEi*92], have demonstrated that this software overhead can be reduced 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, 1.7us) while the network latency is less than 130 cycles (4 us) across a 1024 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 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 represents a key breakthrough, our conclusion from the MPP experience it is not enough to exploit commodity components. You need 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. This requires a resource allocation policy that explicitly preserves interactive performance, while allowing dedicated and unused resources throughout the network to be used by demanding applications: DRAM for memory-intensive programs, disks for I/O bound programs, CPUs for parallel programs.

2.3 Why NOW Now?

The idea of using idle resources over a network has been around nearly since computers became networked, and parallel computing on clusters of workstations is hardly new[MiLi91,Zho92]. What is new is the convergence of technologies and systems concepts that together make NOWs more attractive than ever before.

- *The "killer" network.* Switched local area networks allow bandwidth to scale with the number of processors and low overhead communication protocols have made it possible to do very fast communication over a LAN. These technologies have been proven in MPP systems that span several tens of meters. They are emerging in the LAN arena, with ATM and other recent alternatives, including the Myrinet presented in this issue.
- *The "killer" workstation.* Workstations have become extraordinarily powerful. A top end 1994 workstation is 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 of the time will be spent waiting for I/O. NOWs offer another alternative. A huge pool of memory potentially exists on the network; this memory can be accessed far more quickly than local disk. Furthermore, when I/Os are required, they can be striped across multiple workstation disks, much as in a RAID. The key enabling technology is again the network.

It should be clear that the advantage of a NOW is not just for parallel computing. Rather, it is the opportunity to focus a large collection of resources on a single program: large memory, large disk, or large processing. We believe it is time to concentrate on building large systems out of high volume hardware and software components, and to raise the level at which we do systems research. In taking this research approach, there is the chance to make future high volume components better suited as a building block for such large scale systems.

3. Opportunities for NOW

In this section, we discuss some of the advantages a NOW could offer, when implemented on a building-wide scale of hundreds of machines. The discussion is organized around the pool of resources in a NOW: memory, disks, and processors. In each case, we ask how a NOW system can be more to the end user than simply a bunch of machines on a fast network.

3.1 Memory

Fast network communication makes it attractive to use the aggregate DRAM of a NOW as a giant cache for disk. This has not been practical in the past on Ethernet because it would consume too much of the bandwidth of the shared media and because even on an idle Ethernet, the time to fetch data across the network is only marginally less than a local disk access. On emerging switch-based local area networks, ample bandwidth is available and the remote memory access time is an order of magnitude faster than disk. Table 2 shows a conservative estimate of the time for an 8 KB access on an 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 in Section 4.1 and in [vEi*95]. 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.

	Ethernet		155 Mbit/s ATM	
	Remote Memory	**Remote Disk**	**Remote Memory**	**Remote Disk**
Mem. Copy	250 µs	250 µs	250 µs	250 µs
Net Overhead	400 µs	400 µs	400 µs	400 µs
Data	6250 µs	6250 µs	800 µs	800 µs
Disk	--	14,800 µs	--	14,800 µs
Total	**6,900** µs	**21,700** µs	**1,450** µs	**16,250** µs

TABLE 2. Time to service a file system cache miss from remote memory or disk for Ethernet and for 155 MBit/s ATM (assuming 50% of peak ATM bandwdith).

3.1.1 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 has widened, this illusion has broken down and 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, suggests that programs run 10% to 30% slower using network RAM than if the program fit entirely in local DRAM; using network RAM is 5 to 10 times faster than thrashing to disk.

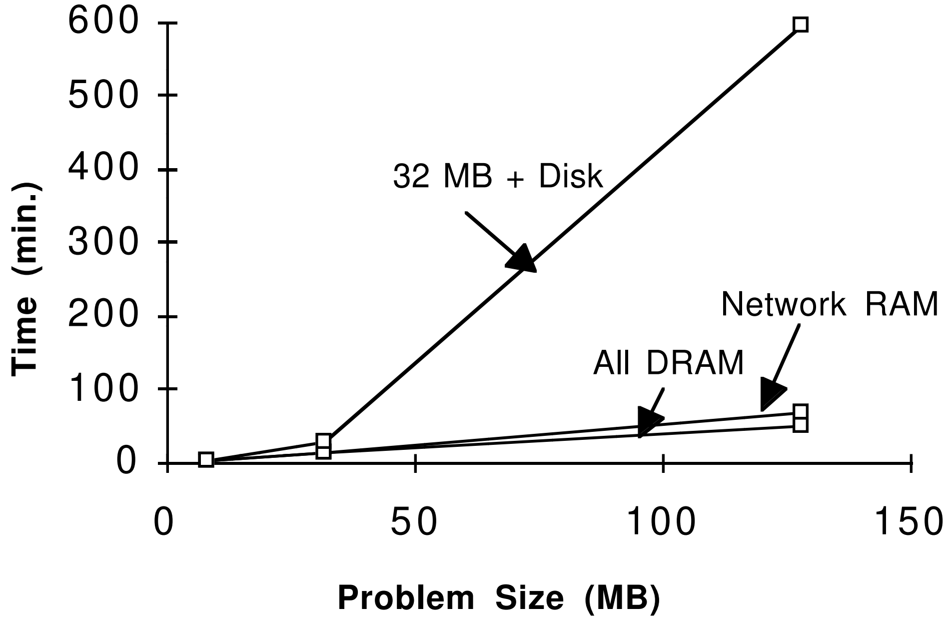

FIGURE 2. Estimated execution time as the size of the multigrid problem increases running on three systems: workstation with 32 MB of DRAM plus disk, one with 128 of DRAM, and one with 32 MB plus paging to DRAM on other machines on the network.

3.1.2 Cooperative File Caching

Analogously, we can improve file system performance by cooperatively managing the file caches on each client workstation. Traditionally, network file systems cache files in local client memory and on client disks to reduce network accesses; they also cache files in memory at the server to reduce disk accesses. In a building-wide NOW, the aggregate client memory far outstrips the memory that can be feasibly put at the server. Cooperatively managing this large client memory has two benefits. First, a number of files, such as executables and font files, are used by more than one client; on a cache miss, these files can be fetched 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.

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 16MB of file cache and the server cache is

128MB, cooperative caching reduced disk reads by a factor of 2, improving file read performance by 80% [Dah*94].

	Cache Miss Rate	Read Response Time
Client-server	16%	2.8 ms
Cooperative caching	8%	1.6 ms

TABLE 3. Impact of cooperative caching: 42 workstations, 16MB/workstation, 128MB/server

3.2 Redundant Arrays of Workstation Disks

RAID (Redundant Arrays of Inexpensive Disks) systems deliver higher bandwidth, capacity, and availability than can be achieved by a single large disk by hooking together arrays of small disks. However, RAID systems have some drawbacks. The cost per byte of disk storage is often worse than single disks by a factor of 2, due to the hardware needed to manage the RAID. Further, the RAID must be connected to a host computer, which is often a performance and availability bottleneck. Although RAID systems use redundant storage to ensure a large MTTF, if the host computer crashes, the RAID becomes unavailable.

NOWs provide the opportunity to 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 workstations on the network. Effectively, the fast network needed for network RAM and cooperative file caching can also be used 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.

3.3 Parallel Computing

NOWs also provide an opportunity to support high-performance parallel applications within an everyday computing infrastructure. For many real-world applications, we need all of: 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 AMES/UCLA chemical tracer model (GATOR) [DeSm94]; it models atmospheric chemistry in the Los Angeles Basin and has been used for detailed air pollution studies.

A model has been developed 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). The predicted wall-clock times for the computation portion of the application have been validated to within 30% 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 ODE phase is highly parallel. The computation involves 36 billion floating-point operations; 3.9 GB of input are needed for the run, and 51 MB of output are produced. We consider a 16-node C-90 (300 MFlops and 10MB/s disk per CPU), a 256-node Paragon (12 MFlops and 2MB/s disk per node), and a number of hypothetical 256-node RS/6000 NOWs (40 MFlops and 2MB/s disk per node). The baseline NOW system assumes Ethernet, PVM, and a sequential file system. The performance of this system is dreadful, taking three orders of magnitude longer than the Paragon or C-90. The performance is limited by sharing a single Ethernet among a large number of high-performance processors. Upgrading to an higher bandwidth ATM network dramatically improves performance of the transport phase, improving overall performance by an order of magnitude. We are still limited by the bandwidth of the sequential file system (2MB/s). Adding a parallel file system that can deliver 80% 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 is competitive 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 is much higher than 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.

Machine	ODE	Transport	Input	Total	Cost
C-90 (16)	7	4	16	27	$30M
Paragon (256)	12	24	10	46	$10M
RS-6000 (256)	4	23340	4030	27374	$4M
“ + ATM	4	192	2015	2211	$5M
“ + Parallel file system	4	192	10	205	$5M
“ + low overhead msgs	4	8	10	21	$5M

TABLE 4. Predicted execution time in seconds for GATOR simulation of 12 hours of weather on a Vector Supercomputer, MPP supercomputer, and 4 hypothetical versions of NOW.

3.4 Workloads of a building-wide system

The measurements above suggest that NOWs can work well as dedicated systems. There is also an opportunity to use the infrastructure in place for interactive work for demanding applications. The key question is whether a NOW can run large programs with the performance as a dedicated large computer and run small programs with the interactivity of a dedicated workstation. To investigate this combination we simulated the impact sequential workstation jobs and MPPs jobs may have on one another[Arp*94].

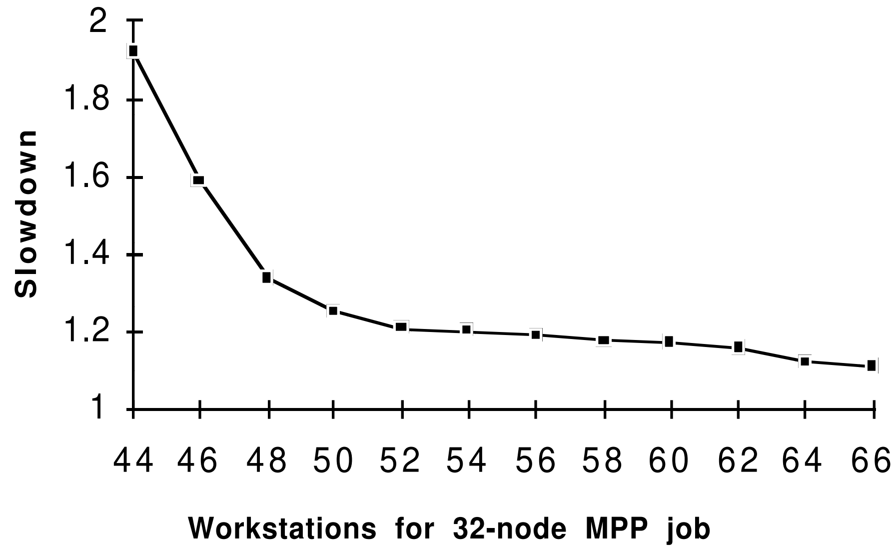

FIGURE 3. Slowdown of a 32-node MPP workload from LANL running on a NOW running a sequential workload as the number of workstations in NOW increases

We collected traces from a local cluster of 53 DECstation 5000/133s with 64 MB 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. Data was collected for two months, resulting in roughly 3000 workstation-days of traces. The workstation traces used in our simulations are randomly selected 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% of workstations are available 100% of the time. (We consider a machine available if there is no user activity or active jobs for one minute.) This result is in direct contrast 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 do you need to run the MPP workload without interfering with the workstation users. Figure 3 shows that for these traces, the parallel workload of a 32-node MPP runs only 10% slower when run on 64 workstations that are running a typical sequential workload as well. This is like getting almost a CM-5 for free.

3.5 Summary of Opportunities

A NOW system offers more than a collection of workstations on a fast network. It provides an opportunity to make advances in traditional system functions, such as virtual memory and file systems, as well as parallel computing. A NOW user's approach to obtaining higher performance may differ from that which is typical for MPPs. First, avoid going to disk by using all the DRAM on the network. If the application is still not fast enough, try using all the disks on the network to speed up the remaining I/O. If it is still not fast enough, parallelize the computational portion. This layered approach seems more attractive than the traditional first step required of MPP users: completely re-write your program before you can see any benefit from the machine.

4. The Berkeley NOW Project

In this section we examine technical challenges inherent in realizing the opportunities of NOW and outline how these are being addressed in the Berkeley NOW project. Our approach is guided by a principle of using commercial off-the-shelf systems wherever possible, in recognition of the rapid advance and tremendous investment in such technologies. As a research vehicle, there are two additional advantages: the implementation is quicker if you avoid re-invention and exploiting mostly off-the-shelf technology will simplify the transfer of new ideas and technology.

4.1 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 communication performance of real programs.

Several studies have examined the communication characteristics of parallel programs and, as in the GATOR example above, many important programs transfer many small messages and are sensitive to communication overhead. What is perhaps more surprising is that many conventional LAN applications exhibit similar characteristics. We obtained a trace of network file system (NFS) traffic over one week from 230 clients of our departmental file servers. Although file transfers are performed in large blocks, we found that 95% 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 returned to the user, so NFS performance is directly coupled to the round-trip message time, i.e., the overhead and latency.

On Sun Sparcstation-10s connected by Ethernet we measure 456 µs of processor overhead plus (unloaded) network latency on a single message and a peak bandwidth of 9 Mb/s through TCP/IP. With the same processors and a Synoptics ATM network the bandwidth increases to 78 Mb/s, but the overhead plus latency also *increases* to 626µs.[1] If we apply these coefficients to our trace, the eight-fold increase in bandwidth reduces

1. Note that these measurements are within 20% of other ATM networks. The network latency component varies for different switches from about 10µs to 100µs depending on the specific configuration. The network interface adapter adds as much as 100µs to the latency, but the largest fraction of the time is the processor overhead resulting from the system software.

the data transmission time component by that amount, but the overall improvement is just 20% 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 10µs. This is technologically feasible, but leaves very little room for compromise. For example, it is equal to 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 in meeting this goal, while others make it quite challenging. In NOW we will have faster processors, but greater constraints on where the network connects into the node. We will have somewhat higher link bandwidth, but may have greater routing delay and less than complete reliability. The nodes support a full Unix system, with relatively rigid device and scheduling interfaces.

The focus of our work is 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. This means that data and control access to the network interface must be mapped 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 is likely to need to assist in supporting message loss as an infrequent case.

One initial prototype is a cluster of HP9000/735 using an experimental “medusa” FDDI network interface that connects to the graphics bus and provides substantial storage in the network interface. As described in a companion paper[Mart94], with user level Active Messages we are able to obtain a processor overhead of 8µs, including support of time-out and retry. This also includes almost 3µs of processing that is entirely an FDDI artifact. The network and adapter latency adds an additional 8µs. We are able to obtain the full link bandwidth for large transfers and obtain half of the peak bandwidth on 175 byte messages, compared to 760 bytes for single copy TCP and 1350 for TCP. Constructing conventional sockets on top of this layer, we see a one-way message time of about 25µs, nearly an order of magnitude faster than TCP or single-copy TCP on the same hardware. Our final demonstration system will utilize either a second generation ATM LAN or a retargeted MPP network, such as the Myrinet[Seit95]. 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 difference in this work, as compared to traditional LAN interfaces, is first the orientation toward low overhead, low latency communication, second, the quality of the interconnect itself and the simplicity that derives from that, and third, the recognition that we have some control over all the nodes that attach to the network and can thus make strong assertions about the endpoints.

4.2 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 is due in part to two significant impediments: implementing global services in the context of existing commercial operating systems and the sociology of global resource sharing.

4.2.1 GLUnix structure

Recall that the hardware argument for NOWs is that large scale computer systems should be built 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 re-implementing 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 is to provide the global services of a NOW by "gluing together" local UNIXs running on each workstation on the network[1]. As much as possible, this Global Layer UNIX (GLUnix) is built as a layer on top of unmodified commercial UNIXs. By leveraging the complete workstation, including the local operating system, we had a working prototype of GLUnix after only three months effort. This layered approach also better lends itself to tracking advances in the underlying commercial system. The challenge, of course, is performance.

The key technology that allows us to layer efficiently on top of existing systems is software fault isolation [Wah*93]. Traditionally, operating system kernels (other than on PC's) use hardware virtual memory to enforce firewalls between user applications. Recent work demonstrates that you 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 be reduced to between 3-7% 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 user-level; this layer catches and translates the application's 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, it is not possible to provide all the global services a user might want with absolutely no kernel changes. Our goal is to look for the minimal set of changes necessary to make existing commercial systems "NOW-ready". One example of this is

1. 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 PC operating systems, such as Windows NT or even DOS.

replacing the kernel communication software with a low overhead implementation. Another is the use of network RAM within the virtual memory system; this can most easily be implemented 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".

4.2.2 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 personal 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 [Arp*94]. 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[TLC85].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 your idle workstation, it will eventually flush out your virtual memory pages and file cache contents. When you return to the workstation, your response time will be visibly slowed as your working set is paged back in from disk.[1] 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 going idle. This is feasible because of the combination of technologies in NOW; with ATM bandwidth and a parallel file system, 64MB 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 any interactive user can be delayed by external processes.

We also need to deliver a large portion of the aggregate capacity of the system 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, has the advantage that no system support is required; however, it leads to unacceptable performance for processes that communicate frequently. For example, Figure 4 shows the slowdown with local scheduling, compared to co-scheduling all processes of a parallel application[Ous82], as the number of competing parallel jobs increases.Two of the applications send many small messages to random processors and, as long as enough buffering exists on the destination processor, the sending processor is not signif-

1. In fact, at DEC SRC where a system was in place to use idle machines for distributed compiles, people in their offices would tap their keyboards periodically simply to keep their memory contents from disappearing!

icantly slowed. Column 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 processor frequently require data from other processors.

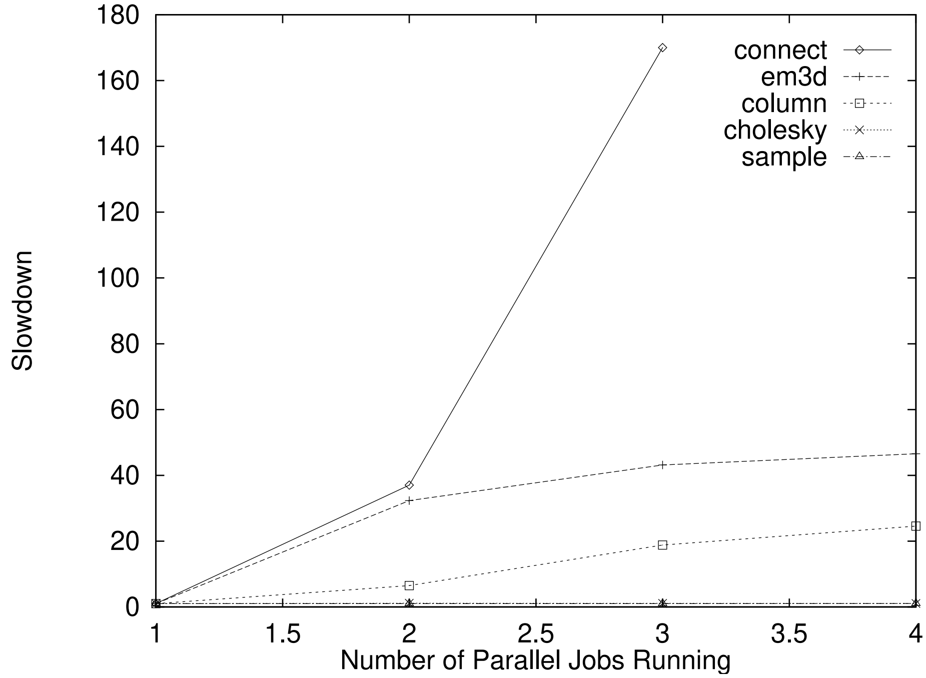

FIGURE 4. Impact of local scheduling on parallel program performance relative to co-scheduling.

Because many parallel programs run as slowly as their slowest process, parallel performance can also be compromised if one of the machines running the parallel job is also being used for interactive computing. Thus, we need to migrate demanding jobs off no longer idle machine 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, so there will usually be a machine to which the evicted process can migrate. For the same reasons, the ability to quickly move processes between machines, along with their memory state, is also important for parallel program performance. While one process is being migrated, the rest of the parallel program is unlikely to be making much progress. The study in Section 3.4 indicates that by implementing fast process migration and choosing idle machines that are likely to stay idle, a typical P-processor parallel workload can be overlaid on 2P interactive workloads 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 non-interactive 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), the entire system must be rebooted. Similarly, the entire multiprocessor must be taken 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 all users would be inconvenienced whenever any small thing goes wrong. Our model is that if a workstation fails, it only affects the programs using that CPU; those programs can be restarted 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. Where tighter security is required, the collection of machines can always be removed from the desktop and placed "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 stand-alone workstation to interactive users by retaining their cached state, and we must provide effective scheduling of parallel applications.

4.3 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. Files are stored on disks at the server, accessible to clients via requests made over the network.

Unfortunately, a central server design has drawbacks in terms of performance, availability, and cost. 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 are sent to the server, ultimately limiting scalability. Furthermore, the DRAM and disk put at the clients do not directly benefit other users. More users can be supported if the file system is partitioned among multiple servers, but this requires the system manager to effectively become *part* of the file system -- moving users, volumes and disks between servers to balance 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 shown in Figure 1, 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 is scalable to an entire enterprise, at low cost.

To achieve this, xFS combines four features not found in other file systems. First, any piece of the file system data, metadata and control can be dynamically migrated between clients and between storage levels; this vastly simplifies both load balancing and failure recovery (any client can take over for any failed client). Second, we use shared-memory multiprocessor-style cache coherence, specifically a write-back ownership protocol, to maximize locality of control and data. Third, 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 Section 3.2). Finally, 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 Section 3.1.2).

5. Conclusion

Computer system design today is dominated by the dramatic rate of advance in small desktop systems, because only these systems offer the large volume and efficiency of production to support a massive on-going 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 the vast collection of resources on the network – processors, memories, and disks – to be viewed as 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 careful attention must be paid 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 indicate that the two kinds of workloads can be combined 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.

References

[Arp*94] R. Arapaci, A. Dusseau, A. Vahdat, T. Anderson, and D. Patterson, “The Interaction of Parallel and Sequential Workloads on a Network of Workstations”, submitted for publication.

[Broo92] E. D. Brooks III, “Massive Parallelism Overcomes Shared-Memory Limitations,” Computers in Physics, Mar 1992, no. 2, pp. 139-45.

[Cul*93] D. Culler, R. Karp, D. Patterson, A. Sahay, K. Schauser, E. Santos, R. Subramonian, T. von Eicken, "LogP: Towards a Realistic Model of Parallel Computation," *Proc. 4th ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming,* 1993.

[Dah*94] M. Dahlin, R. Wang, T. Anderson, and D. Patterson, "Cooperative Caching: Using Remote Client Memory to Improve File System Performance," Proc. of the First Conference on Operating Systems Design and Implementation, Nov., 1994.

[DeSm94] J. Demmel and S. Smith, "Parallelizing a global atmospheric chemical tracer model",Symposium on High Performance Computing and Communications, May 1994.

[Mart94] R. Martin, "HPAM: An Active Message Layer for a Network of HP Worksations," Hot Interconnects II, Aug. 1994.

[MaLi91] M. Mutka and M. Livny, "The Available Capacity of a Privately Oned Workstation Environment," Performance Evaluation, 12(4):269-84, Jul 1991.

[PaHe90] D. A. Patterson and J. L/ Hennessy, "Computer Architecture: A Quantitative Approach," Morgan Kaufmann Pub. Inc., 1990.

[Ous82] J. Ousterhout, "Scheduling techniques for concurrent systems", *Proc. 3rd International Conference on Distributed Computing Systems*, pp. 22-30, Oct. 1982.

[Seit95] C. Seitz, "Myrinet – A Gigabit per second Local-area Network," this issue.

[TLC85] M. Theimer, K. Landtz, and D. Cheriton, "Preemptable Remote Execution Facilities for the V System," in Proc. of the 10th ACM Symposium on Operating System Principles, pp 2-12, Dec 1985.

[vEi*92] T. von Eicken, D. Culler, S. Goldstein and K. Schauser, "Active Messages: a Mechanism for Integrated Communication and Computation," *Proc. 19th Annual International Symposium on Computer Architecture*, pp. 256-267, May 1992.

[vEi*95] T. von Eicken, A. Basu, and V. Buch, "Low Latency Communication over ATM Networks using Active Messages," This issue

[Wah*93] R. Wahbe, S. Lucco, T. Anderson and S. Graham. "Efficient Software-Based Fault Isolation." Proc. Fourteenth ACM Symposium on Operating System Principles, Dec. 1993, pp. 203-216.

[Zho*92] S. Zhou, J. Wang, X. Zheng, and P. Delisle, "Utopia: A Load Sharing Facility for Large, Heterogenous Distributed Computing Systems," Technical Report CSRI-257, University of Toronto, 1992.

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 endebted to Terry Lessard-Smith, Bob Miller, and Eric Fraser for terrific administrative and technical support.

The NOW project has received support from the Advanced Research Projects Agency (#N00600-93C-2481), the National Science Foundation (CDA-9401156), and the California Micro program. Tom Anderson and David Culler are supported by NSF Presidential Faculty Fellowships. The project has received valued support from SUN Microcomputer Corp., Hewlett-Packard, IBM, Digital Equipment Corp., Intel Corp., Thinking Machines, Synoptics, Cisco, Xerox, AT&T , Siemens, Fujitsu, and Exabyte.