Remote Memory Access in Workstation Clusters

Ben Verghese
Mendel Rosenblum

Technical Report: CSL-TR-97-729

July 1997

This work was supported in part by ARPA contract DABT63-94-C-0054.
Mendel Rosenblum is partially supported by a National Science Foundation Young Investigator award.

Remote Memory Access in Workstation Clusters

Ben Verghese and Mendel Rosenblum

Technical Report: CSL-TR-97-729

July 1997

Computer Systems Laboratory
Department of Electrical Engineering and Computer Science
Stanford University
Stanford, CA 94305-2140
{pubs}@cs.Stanford.EDU

Abstract

Efficient sharing of memory resources in a cluster of workstations has the promise of greatly improving the performance and cost-effectiveness of the cluster when running large memory-intensive jobs. A point of interest is the hardware support required for good memory sharing performance. We evaluate the performance of two models: the software-only model that runs on a traditional distributed system configuration, and requires support from the operating system to access remote memory; and the hardware-intensive model that uses a specialized network interface to extend the memory system to allow direct access to remote memory. Using SimOS, we do a fair comparison of the performance of the two memory-sharing models for a set of interesting compute-server workloads. We find that the software-only model, with current remote page-fault latencies, does not provide acceptable memory-sharing performance. The hardware shared-memory system is able to provide stable performance across a range of latencies. If the remote page-fault latency can be reduced to 100 microseconds, the performance of the software-only model becomes acceptable for many, though not all, workloads. Considering the interconnection bandwidth required to sustain the software-only page-level memory sharing, our experiments show that a gigabit network is necessary for good performance.

Key Words and Phrases: CC-NUMA, memory sharing, NOW, workstation cluster, remote memory.

1.0 Introduction

Efficient sharing of resources in workstation clusters has received much recent attention because of the advances in high performance interconnection network technology. New distributed file systems that support more efficient coupling of disk storage have been proposed [ADN+95], as well as new studies of CPU sharing mechanisms, such as remote execution and process migration [Arp+95]. The low latency and high bandwidth of the interconnection networks have also prompted interest in sharing of memory resources. The increasing memory requirements of compute-intensive applications and the large amount of "idle" memory in a workstation cluster is driving system designers to consider schemes to efficiently utilize the total memory in a cluster, possibly by a single workstation. Such a scheme would be attractive because it could greatly improve the performance and the cost-effectiveness of a cluster, while simplifying the cluster-wide resource-allocation problem.

Much of the work on memory-sharing has focussed on mechanisms and policies for utilizing the "idle" memory of the cluster as an alternative to swapping to disk [FMP+95] [ACP+94] [DWA+94]. Given high disk latencies and the speed of current fast interconnects, getting data from a remote node's memory instead of a slow magnetic disk has been shown to be a big win. In this study we are interested in a different comparison; can remote memory be used as an extension of local memory, not just as a disk cache? Does the performance of current memory-sharing solutions support this model, i.e., how do these solutions compare to just having enough local memory.

Although memory-sharing solutions are similar in principle, their implementations can be quite different. One area of interest is the hardware support required for access to remote memory. For this study we choose two approaches that represent opposite ends of the hardware support spectrum. The Berkeley NOW proposal [ACP+94] represents a software-only approach that uses system software extensions to support remote memory. This solution is attractive because it potentially requires only commodity hardware, including interconnects and network interfaces. In this solution, the operating system uses the interconnection network to transfer page-sized blocks of information to and from the memories as needed by the execution of the workload.

At the opposite end of the spectrum from the NOW proposal is the hardware-based distributed shared-memory model of the CC-NUMA architecture. In a CC-NUMA machine, the interconnection network enters the system at the memory bus. The specialized network interface logic used is able to fetch cache lines from remote memory when required. Some examples of CC-NUMA machines from academia are the Stanford FLASH [Kus+94], MIT Alewife [ACD+91], and the Wisconsin Typhoon [RLW94], and from industry are the Convex Exemplar, Sequent STiNG, and SGI Cray Origin. Though the above systems are all tightly coupled multiprocessors, the same hardware model can be used to connect clusters of workstations too, e.g. the distributed FLASH proposal [Kus+94] and the SUN s3.mp [NAB+94].

In this study, we do a performance comparison between the different memory-sharing models using SimOS [RHW+95], a complete machine simulator, and a set of relevant workloads. SimOS allows for a fair comparison of the models by providing identical simulated hardware, except for differences required by the memory-sharing models. Based on our results, a software-only model

is unable to provide the necessary memory-sharing performance. If in the future the page-fault latency can be lowered to the 100 microseconds range, the performance of the software model becomes acceptable for many, though not all, workloads. In contrast, the hardware shared-memory model provides stable performance for a range of remote cache-miss latencies.

Section 2.0 gives a motivation for using memory sharing and describes the two memory-sharing models. Section 3.0 explains our experimental environment, including the simulator support for implementing the memory-sharing models, and the workloads. In Section 4.0, we present experimental results from running the workloads with each model, and analyze their performance based on the memory-access characteristics of the different workloads. Section 5.0 analyzes the possible performance of a future optimistic NOW model. Section 6.0 discusses other factors that may affect the performance of the two memory-sharing models, Section 7.0 discusses related work, and Section 8.0 concludes.

2.0 Memory-sharing Models

Memory-sharing in workstation clusters is motivated by two different possible uses of these clusters. In the more traditional workstation-based distributed systems environment, memory-sharing enables more effective use of the available memory. Rather than having each user's workstation have enough memory to accommodate the largest possible job, memory sharing allows a workstation to utilize the "idle" memory of other users' workstations to run large jobs, without excess slowdowns caused by paging to disk. Workstations in this configuration are physically distributed on users' desks, and must also continue to support the smaller interactive jobs such as editors and window servers while crunching on the large resource intensive jobs. Since the memory of modern workstations can account for 25% to 50% of the machine's price, the ability to get by without stacking every machine with excess memory makes this an attractive proposal.

In the second environment, memory sharing is used to construct "compute servers" from a cluster of commodity workstations. This proposal differs from the traditional workstation-based distributed system in that the machines can be physically co-located in the same machine room, and can support a workload of resource-intensive jobs submitted by a collection of users. The system software attempts to balance the jobs across the cluster's machines with a goal of maximizing throughput. Again, memory-sharing allows for efficient execution of jobs with memory requirements larger than a single workstation's memory, and simplifies the resource-allocation problem.

The two environments have implications for both the interconnect technology used and the types of workloads supported. The wide physical distribution of the nodes in the user-workstation environment suggests that commodity local-area interconnect technology will be used to connect these machines. Wire transmission delays and limited bandwidth will cause the communication to be slower compared to the interconnect used to communicate between the co-located workstations of the "compute-server" environment.

The workloads being supported in the two environments also differ. The load balancing of compute-intensive jobs of multiple users on the "compute-server" cluster provides for the possibility of having one or more unrelated processes on a single workstation. This enables the

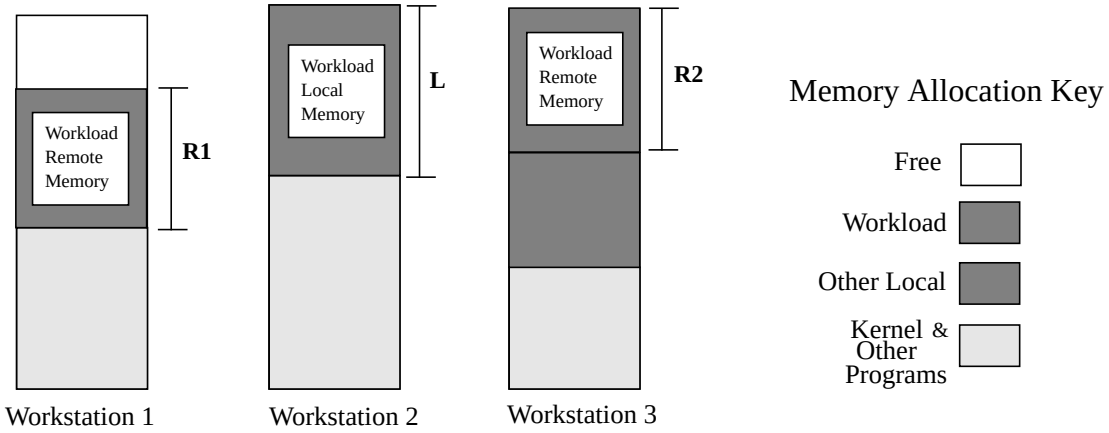

FIGURE 1. Memory-sharing example showing memory allocation in a three workstation cluster. A workload running on workstation 2 is using remote memory on workstations 1 and 3 in addition to local memory on workstation 2. The total memory requirement of the workload is (L + R1 + R2), with (R1 + R2) being allocated remotely. A part of local memory has been allocated to the kernel and other processes.

system to potentially hide the latency of communication for accessing remote memory by overlapping it with useful computation. This throughput enhancing technique is less applicable to the single-user workstation environment.

Both environments require the same basic model of execution where remote memory is viewed, both conceptually and through performance, as a transparent extension of local memory. Figure 1 shows the type of memory sharing we study in this paper. Workstation 2 is running a workload whose memory requirement exceeds the available local memory. Rather than paging to disk, the memories of other workstations in the cluster are used to hold the additional pages needed by the workload. We assume that there is always sufficient remote memory available.

Just as the virtual memory abstraction of modern machines is implemented by a combination of hardware and operating system support, remote memory access requires such support as well. We study two memory-sharing models. The NOW model uses a software-only approach, and requires extensive support from the operating system, while the CC-NUMA model uses a hardware-intensive approach. Both the CC-NUMA model and the NOW model are capable of providing memory-sharing in either of the environments. Figure 2 shows the access mechanisms to local and remote memory in the two models. The following sections present these models in detail.

2.1 The NOW Memory Model

In the NOW model for sharing memory, each node is a workstation with a processor, local memory, and an interface to a high-speed interconnect such as an ATM or a Myrinet switch. To avoid changes to the workstation hardware, access to the network is through the standard interface provided by the I/O bus. Memory-sharing is implemented by having the operating system use remote memory as a fast paging disk.

In the NOW model, an access to a page not in local memory results in a page fault exception. The OS fault handler determines that the page is in the memory of another workstation, and sends a request for the page over the network. At the remote workstation, the kernel services this request by locating the requested page and shipping the data back. This sharing model has been recently

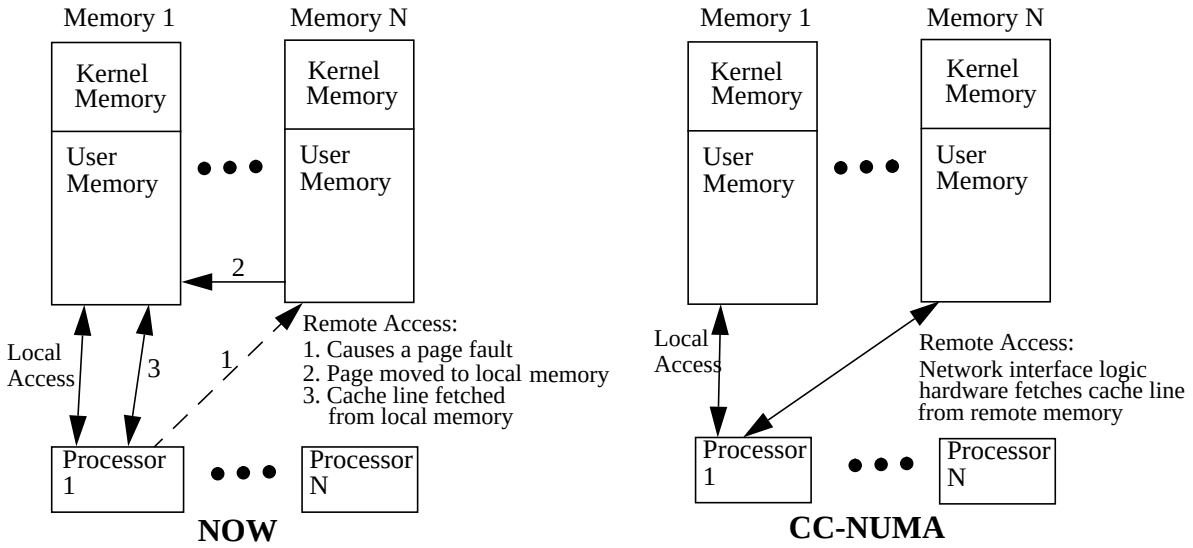

FIGURE 2. The access mechanisms to local and remote memory for the two memory-sharing models. A cache miss to local memory leads to direct access of the data in both models. The same is true for a cache miss to remote memory in the CC-NUMA model, since the hardware does all the work transparently. For a cache miss to remote memory in the NOW model the OS needs to fetch the page from a remote memory.

used in proposed schemes such as network RAM [ACP+94], global memory manager (sharing of memory pages) [FMP+95], and co-operative caching (a global file system buffer cache) [DWA+94].

The latency of a remote page-fetch can be quite high, therefore the faulting user process is descheduled for the duration of the page fetch, and another process is scheduled, allowing communication to be overlapped with computation. When the remote page is received by the requestor, it is copied from the network interface into local memory using DMA, and the OS is informed through an interrupt that the page is ready. The OS then puts the faulting process back on the run queue, and this process finds the page in local memory when it runs next. Another page will potentially have to be evicted from local memory to free up space.

In this model the most important factor is the page-fault latency for fetching a page (4 Kbytes) from the memory of a remote node. We explore the performance impact of various values for this latency. Anderson et al [ACP+94] have demonstrated that the latency for a page fetched from remote memory over an ATM network is about 1 to 1.5 milliseconds. We will use 1 millisecond as the default page-fault latency for the NOW architecture. Additionally, Jamrozik et al [JFV+96] showed that by using sub-pages, the first sub-page containing the referenced data could be received substantially faster. Based on the sub-page scheme, we will also study a page-fault latency of 0.5 milliseconds.

2.2 The CC-NUMA Memory Model

In the hardware-intensive approach of the CC-NUMA model, the interconnection network is attached directly to the memory bus through specialized network interface logic. This model provides transparent access to remote memory by having the interface snoop the cache miss requests coming from the processor, and recognize requests for cache lines that are on remote

nodes. For these cache lines, the hardware sends a message to the remote node requesting the data.

In addition to shipping data bytes back and forth, the network interface logic has to track the processors caching each line, and maintain cache coherency. DASH[LLG+92], Alewife[ACD+91], and Typhoon[RLW94] are existing examples of these systems, which are referred to as CC-NUMA (cache-coherent non-uniform memory architecture) multiprocessors. This model allows the operating system and user applications to treat the memory of the workstations as they would that of a shared-memory multiprocessor.

The tight coupling of the network interface with the memory system, and the high-speed processing of the hardware communication engine allows for very low latency communication. The low overhead means that the total communication time can be dominated by the wire transmission delays. For the compute server model, such as the Stanford FLASH machine[Kus+94], a remote access stalls the processor for around 1200 nanoseconds. During this time the processor is limited in its ability to overlap computation with the communication (Newer processors do not necessarily have this limitation, Section 6.1).

A variation on this model has the workstations physically distributed on users' desks, and transmission delays result in higher communication latencies. Such CC-NOW (cache-coherent network of workstations) clusters [Kus+94] [NAB+94] are estimated to have remote-memory access times around 3000 nanoseconds.

2.3 Qualitative Comparison of the Models

Given the differences between the two memory models, it is difficult to say which model will perform better for different workloads. The NOW model pays a much higher communication cost (0.5 - 1 milliseconds vs. 1.2 - 3 microseconds) yet it receives more data (4 Kbyte page vs. 128 byte cache line), and has the additional advantage of being able to overlap computation with the communication. The performance benefits of these features depend heavily on the application workload. For example, if the application's locality of reference corresponds to accessing entire pages, fetching whole pages may have an advantage over repeated cache misses to fetch the page. Similarly, if additional processes are available in the workload to overlap computation with the remote page fault processing, the NOW model can do useful work while the CC-NUMA processor would be stalled.

3.0 Experimental Environment

To compare the two different models for remote memory access we use SimOS [RHW+95], a complete machine simulator. SimOS models the processor, caches, memory, I/O devices, and interconnection network of a node in a workstation cluster in sufficient detail to run commercial operating systems and applications. Using SimOS has a number of advantages over attempting this study on a real system. On real machines, a fair comparison between memory models would be difficult because of the differences in system architectures and operating systems. SimOS provides a common platform for comparing the memory models in otherwise identical environments. A real implementation would give us only one point in the parameter space, and

would make it cumbersome to explore the performance of interesting variations on the basic models. SimOS provides the flexibility to change memory-sharing models and their parameters.

For both memory-sharing models, we simulate the same high-end workstation representing a node in the cluster. The system parameters of the node were chosen to be representative of a workstation that might typically be found on a desktop today. Each node includes a 300Mhz MIPS R4000-like processor with 32K on-chip instruction and data caches, one megabyte 2-way unified secondary cache with a 300 nanoseconds miss latency to local memory and a 128 byte cache line size, and several common I/O devices, including SCSI disks with timing based on a HP 97560 disk.

For the CC-NUMA case, we configured SimOS with a NUMA memory system model. The NUMA memory system models the local bus, the latency and occupancy of the local and remote network interface logic, the interconnection network, and the DRAM access time. Pages are allocated to local memory on a first-touch basis until local memory is filled. Once local memory is filled, pages are allocated from remote memory. This is a static allocation scheme, and NUMA optimizations such as page migration and page replication are not used. The processor stall time for cache misses is dependent on whether the page is located in local or remote memory.

Implementing the NOW model proved to be much more interesting because of the large operating system component. The obvious approach would have been to extend the kernel to fully support one or more of the proposed NOW implementations. This would have resulted in fixing the relatively large software overheads in the client and the server as seen in the current NOW proposals, and limiting our flexibility to investigate the effect of future more aggressive NOW implementations and interconnection networks. On the other hand, it was also not possible to simply have the entire effect of the NOW model configured as a parameter in the simulation, as was done for the CC-NUMA model. The operating system needed to be aware of the remote-memory page fault, in order to support the overlap of communication and computation by possibly scheduling another job during the communication latency. This implies functionality that wasn’t present in the operating system.

We use a novel hybrid approach, extending the kernel to implement some of the functionality required for the NOW model, and implementing the rest of the necessary functionality in SimOS. We extended the OS to “know” about remote pages. A reference to these pages by the workload causes a remote-memory page fault that trapped into the operating system. This page-fault handler deschedules the faulting process so other processes can be run. When the page is received from the remote node, SimOS generates an interrupt to inform the OS. The handler for this interrupt reschedules the faulting process. These trap, interrupt, and scheduler overheads seemed unavoidable for any NOW implementation that supports the overlapping of communication and computation.

All the other overheads for handling remote memory access in a NOW — formatting and sending the request for the remote page, all round-trip network timing, network bandwidth effects, and service time on a remote node — are modelled as a remote-memory page-fault latency in the SimOS simulator. This gave us the ability to evaluate the NOW model over a range of latencies. SimOS also maintains complete usage information for the local pages, and uses this to implement a perfect LRU page replacement policy.

TABLE 1. Description of the six workloads used in the study. The first three are multi-programmed and the next three are single applications.

Workload	Description	Memory Utilized (Mbytes)
PMAKE	Multiprogrammed - Two parallel makes of the gnuchess program	61.3
VCS	Multiprogrammed - Two Chronologic VCS verilog simulations	25.4
SPLASH	Multiprogrammed - Ocean and Raytrace from SPLASH	73.0
DB	Sybase database server with TPC-B type queries	43.2
SIMOS	Simulation of a 4 processor system on SimOS	42.3
DELAY	CAD tool for gate delay and timing	78.9

Whereas existing academic and commercial systems have demonstrated NUMA machines that have similar overheads to our CC-NUMA model, the NOW simulation model needs to be considered as optimistic. Existing systems have shown considerable overheads for network communication, protocol processing, page replacement, etc. Our NOW models assume that these overheads can be made quite small. For example, in the NOW model we assume that by using sub-pages, the whole page can be received in 0.5 milliseconds. Actually only the first one Kbyte can be received in 0.5 milliseconds, and the whole page would take about one millisecond, possibly causing additional faults.

SimOS is used to boot a commercial operating system, SGI's IRIX 5.3, and run representative compute-server workloads. The fast simulation mode in Simos is used to run each workload till all initialization is done. At this point the memory and global file cache is "warm". The subsequent "interesting" part of the workload is then simulated in detail.

In picking workloads for the study, we had two goals. First, we wanted applications that would be representative of large jobs that would run on clusters of workstations. Second, we wanted workloads that represent the two possible environments for memory-sharing described in Section 2.0; single applications to represent the user-workstation environment, and multiprogrammed workloads for the "compute-server" environment. We use six compute-intensive workloads for this study as shown in Table 1, three multiprogrammed and three single large applications. These workloads represent applications from different disciplines — program development, circuit simulation, gate-delay computation, scientific applications with graphics, system simulation, and databases.

4.0 Experimental Results

To compare the memory-sharing models we configured SimOS to model the four configurations: NOW (one millisecond page-fault latency), SUBPAGE (NOW model with 0.5 millisecond page-fault latency), CC-NUMA (1.2 microsecond remote cache-miss latency), and CC-NOW (NUMA model with 3 microsecond remote cache-miss latency). We simulated the execution of the workloads on the memory-sharing configurations with differing amount of local memory

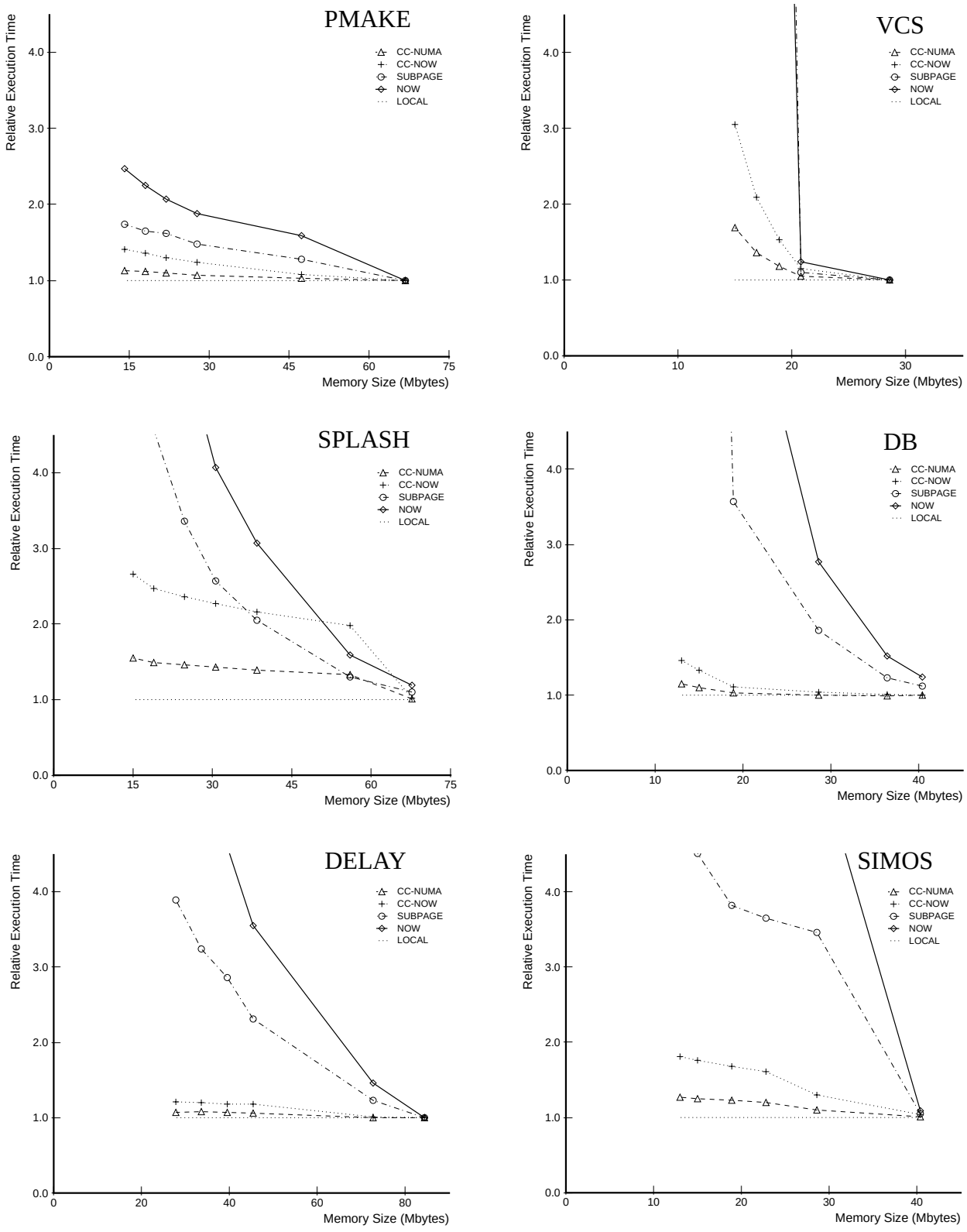

FIGURE 3. Relative execution time from actual runs as a function of local memory size. Results are shown for four configurations, CC-NUMA, CC-NOW, SUBPAGE, and NOW. The execution times are shown relative to the case where all memory is local, which is 1 (LOCAL).

available to the workload. By varying the amount of local memory, we can explore the ability of the memory-sharing configurations to use the memory of other workstations.

Figure 3 shows the workload execution time for each workload and memory-sharing configuration plotted against the available local memory size. The execution time shown is relative to the unlimited local memory case (LOCAL) where no remote memory is being used. We truncated results above a seemingly unacceptable four times slowdown over the LOCAL case.

Figure 3 clearly shows that the OS-based memory-sharing architecture (**NOW** and **SUBPAGE**) results in poor performance for all the workloads as local memory decreases. The performance is competitive only when the local memory size approaches the total memory footprint of the workload. In this limiting case remote memory is not being used. The faster SUBPAGE model improves performance over the NOW model, but still results in large slowdowns when compared to having enough local memory.

A closer look at the NOW runs with smaller local memory shows that a significant increase in execution time is caused by large increases in the idle time. This idle time results from the workload's processes being descheduled while pages are faulted in from remote memory. While these NOW models provide a significant performance improvement when compared with paging to a disk, they are still too slow to replace having enough local memory.

The CC-NUMA model, on the other hand, provides consistently good performance for the workloads. The performance for CC-NUMA is close to that of LOCAL even for small local memory sizes. Unlike the NOW and SUBPAGE models, CC-NUMA performance degrades gracefully as the local memory is reduced. If the remote-cache-miss latency is increased to 3 microseconds (CC-NOW), the absolute performance decreases. However, this does not change the nature of the performance profiles.

The size of local memory used, at the low end of the range of runs, may seem small compared to workstation memories. However, we are only showing memory available to the workload, kernel code and initial data. As explained in Section 2.0, significant additional local memory is normally used on workstations by the kernel and other programs, such as the X-server and editors, and this memory is then not available to the workload. Also, we have chosen complex applications with interesting data sets, however in real life they could use significantly larger data-sets requiring much more memory.The behavior of these workloads, as available local memory is reduced, should be representative of larger programs too.

4.1 Analysis of Results

For the OS-based (NOW) model, the PMAKE workload shows the best performance. The performance of the SPLASH and DELAY workloads comes next. The performance of the VCS, SIMOS, and DB workloads degrades rapidly when local memory is reduced. To understand the reasons behind the observed performance, we configured SimOS to generate a cache miss trace that was then used to characterize the memory-access and cache-miss patterns of the workloads. The results are show in Figure 4 and Table 2.

Figure 4 shows two graphs for each workload. The first graph shows the size of the memory working-set over the execution of the workload. The different lines show the memory accessed

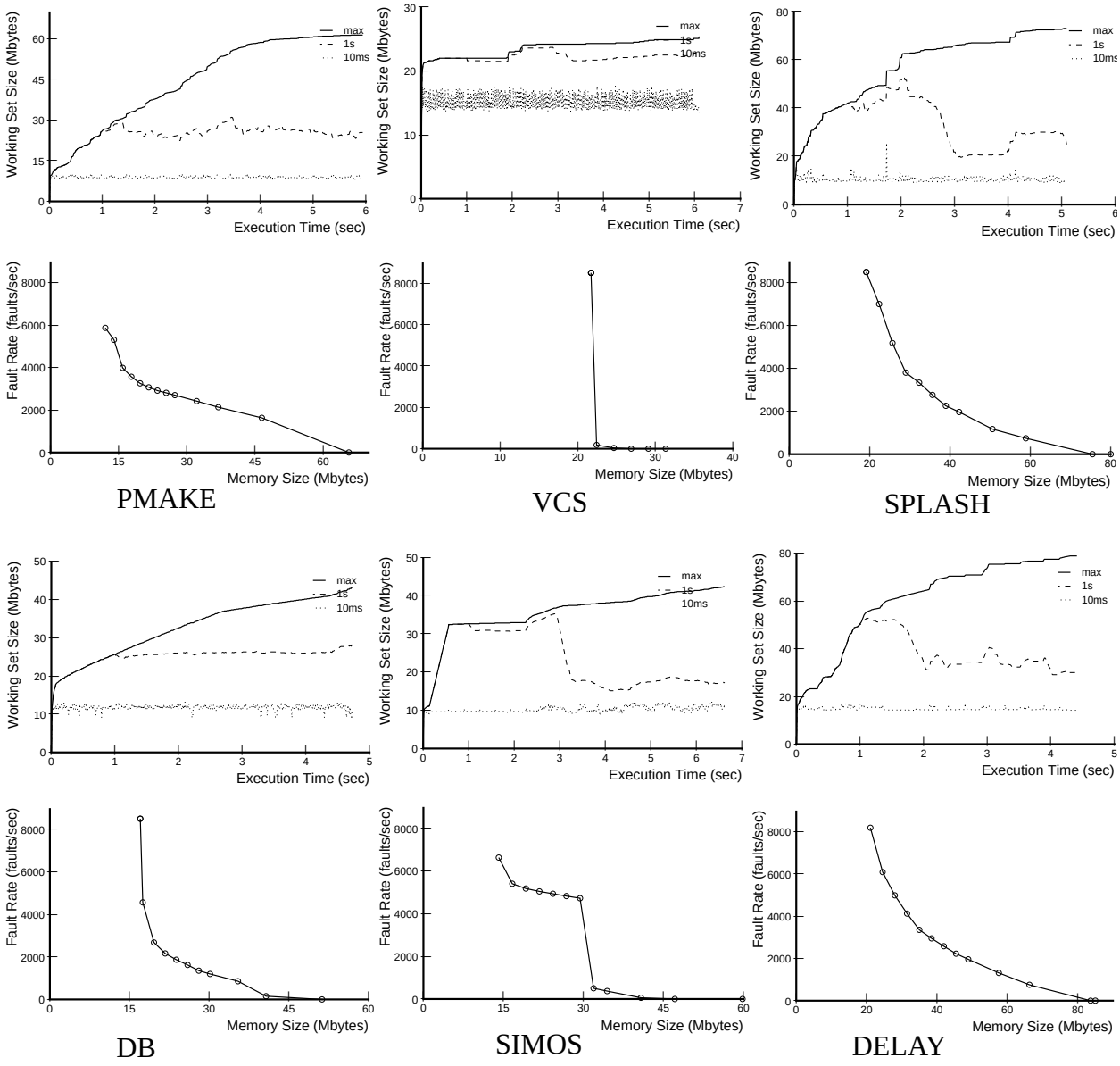

FIGURE 4. Working set sizes and fault rates for the six workloads. There are two graphs for each workload, the upper shows the working set size versus time and the lower shows the fault rate versus memory size. For the working set graph there are three lines corresponding to three window sizes 10 milliseconds (10ms), one second (1s), and the full run (max). The fault rates are for an LRU policy. These graphs are generated using a trace of cache misses.

over short intervals of 10 milliseconds and relatively long intervals of one second, and the change in the total memory footprint of the workload. The second graph shows the average required fault rate as a function of available user memory. A perfect LRU page replacement policy was used in determining the fault rate, and we assumed that local memory was warmed with pages from the application's memory footprint on a first-touch basis. Both graphs include the fixed initial part of memory used by the kernel for its code and initial data.

Table 2 shows the cache-miss characteristics for the workloads: cache-misses/1000 instructions for the 1Mbyte cache size that we use in our experiments, and for a 512Kbyte and a 4 Mbyte

TABLE 2. Cache miss characteristics for the workloads. The left three columns show misses per 1000 instructions (non-idle) for each workload for three different cache sizes, 512Kbytes, 1Mbyte, and 4Mbytes. The right two columns show percentage of misses to local memory for two sizes of local memory. For each workload, "Small" is the memory size of the second point from the left on the CC-NUMA curve in Figure 3, and "Large" is that for the second point from the right on the same curve.

Workload	Cache Misses (per 1000 instr.)			Percentage of misses from local memory	
	512 Kbyte	1 Mbyte	4 Mbyte	Small Mem.	Large Mem.
PMAKE	9.1	5.9	3.2	55%	90%
VCS	257.0	210.0	80.0	74%	97%
SPLASH	68.4	58.1	28.5	18%	46%
DB	19.0	8.7	2.5	75%	99%
SIMOS	25.8	14.9	7.7	36%	75%
DELAY	6.2	1.7	1.1	38%	97%

cache for comparison; and the percentage of cache misses that were satisfied from local memory for two different sizes of local memory.

We now describe three important workload characteristics, and show how they affect the performance of the workloads.

4.1.1 Page-fault Rates

From Figure 4, the page-fault rate curves provided a good predictor for the performance of the NOW model. A low page-fault rate means there is little communication with remote memory while a high page-fault rate seriously degrades the performance because the processor will potentially be idle during the page-fault service time. This difference can best be seen in the fault curves for the PMAKE and VCS workloads.

The relatively shallow page-fault curve of the PMAKE workload means that the remote page fetches performed by the NOW model will only gradually increase as the amount of local memory decreases. Furthermore, the workload has good spatial locality so in each ten millisecond window only few pages are accessed. The end result is the relatively good performance for NOW.

At the other end of the spectrum is the VCS workload. It has a very sharp knee in the page-fault curve, which represents an important working set of the workload. The working-set graph shows that most of the pages in the memory footprint are touched frequently, and the accesses to pages are quite random in the short term. As a consequence, if the available local memory is reduced below 20 Mbytes the page-fault rate increases drastically. This explains the huge drop in the NOW model's performance when local memory dropped below this threshold.

The other workloads are between these two extremes. The SPLASH and DELAY workloads exhibit page-access locality, and fault curves similar to PMAKE. The DB and SIMOS workloads have fairly steep fault curves similar to VCS because they show more random accesses to pages.

4.1.2 Multiprogramming

The shape of the fault curve also helps explain the inability of the OS-based models to overlap communication with computation through multiprogramming, for some of the workloads. For workloads with a gradual rise in the fault curve, such as PMAKE, it is possible to give some memory to additional processes without causing large increases in the fault rate. The execution of the additional process can be used to hide all or part of the page-fault latency. This effect can also be seen to a lesser extent in the SPLASH workload.

The VCS workload demonstrates the lack of applicability of this technique to hide idle time for workloads with steep fault curves. In such workloads, decreasing the memory available to a process to make room for another process, can increase the fault rate to such an extent that any benefit from overlapping computation with communication is swamped by the increase in idle time.

The hardware-based model will not see any benefit from multiprogramming, if anything it will degrade performance by causing additional cache misses due to interference by the multiple applications.

4.1.3 Cache-miss Rates

The performance of the hardware-based models (CC-NUMA and CC-NOW) is not dependent on page faults, but is dependent on the number of cache misses to remote memory because the processor is stalled for the duration of a cache miss. The remote-memory stall time is dependent on two factors, the cache-miss rate of the application and the fraction of misses that go to remote memory. A high cache-miss rate will degrade the performance of these models. For our experiments, we do static first-touch placement of pages in local memory. Therefore, the fraction of misses to remote memory is a combination of the amount of local memory, and the effectiveness of first-touch at placing the appropriate pages in local memory.

Table 2 helps explain the behavior of the VCS and SPLASH workloads, the two workloads that show the greatest performance degradation for the hardware models. The VCS workload has an extremely high cache miss rate for the one megabyte cache used in the study. As the amount of local memory is reduced, the fraction of cache misses to remote memory increases, resulting in additional processor stall time that hurts performance. The longer latencies for CC-NOW exacerbate this problem resulting in the large slowdowns for small local memory sizes seen in Figure 3.

The SPLASH workload has a high cache-miss rate, and suffers from poor initial layout of pages in local memory. The first-touch policy resulted in a majority of cache misses going to remote pages even in the large local memory configuration. The combination of these effects caused the large slowdowns seen by the hardware models. It is particularly noticeable in the CC-NOW model which performs worse than the OS-based models for some local memory sizes. The OS-based models always copy remote pages to local memory so their memory-sharing performance is independent of the workload’s cache-miss rate.

The other workloads have low cache-miss rates or good locality or both, allowing the hardware-based approach to perform well across the range of local memory sizes. Larger caches size will reduce the cache-miss rate, and so improve the performance of the workloads for the hardware model. Moving from an 1 Mbyte to a 4 Mbyte cache reduces the miss rate by 50% or more for all the workloads.

5.0 Optimistic NOW Models

The results from the previous section showed that a NOW model, based on parameters from currently existing implementations, performs poorly when local memory is limited. We decided to investigate whether the performance of the NOW model can be made competitive with the case of having enough local memory, and what the page-fault latency would need to be to achieve this performance.

We assumed an extremely fast page-fault mechanism, and cranked down the page-fault latency to 100 microseconds. Transferring a 4 Kbyte page in less than 100 microseconds requires bandwidth that is significant compared to that of available commodity interconnection networks. This was not an issue for the base NOW and SUBPAGE models because the page-fault latency was much higher. Therefore, for smaller latencies the NOW model incorporated network queuing to account for limited bandwidth, resulting in two new models, the MYRNOW and the ATMNOW. These are projections of how fast the NOW model needs to be, and there are no current systems known to us that are close to this page-fault latency.

The MYRNOW model assumes a gigabit style high-end network modelled on the Myrinet [Bod+95]. The uncontended latency for requesting a four Kbyte page is 100 microseconds split into two components, 20 microseconds of actual latency for a zero-size page, and 80 microseconds of transmission latency, assuming an actual unidirectional data throughput of 400Mbits/s.

The ATMNOW model is based on a currently available ATM-speed network. The uncontended latency is about 220 microseconds; 20 microseconds of actual latency for a zero-size page, and 200 microseconds of transmission latency, assuming a data throughput of 160Mbits/s. In both these models, the 20 microseconds of latency includes two end-to-end network latencies, and the time for the remote node to accept the request and find the requested page.

Figure 5 shows the relative execution times for the workloads for four different models — the new MYRNOW, and ATMNOW models and the previously considered hardware-based models, CC-NUMA and CC-NOW. The workloads can be divided in three categories based on their performance.

For the PMAKE and SPLASH workloads, the performance of the MYRNOW model is within the acceptable range. In fact, it actually performs better than the CC-NOW model, though not the CC-NUMA model. Both of these workloads are multiprogrammed, and have shallow page-fault curves. The idle time is less because of the smaller page-fault latency in the MYRNOW model, and it is able to hide much of this idle time by overlapping computation with communication. The SPLASH workload also has a poor distribution of pages in local and remote memory for the hardware models, as explained in Section 4.1.3.

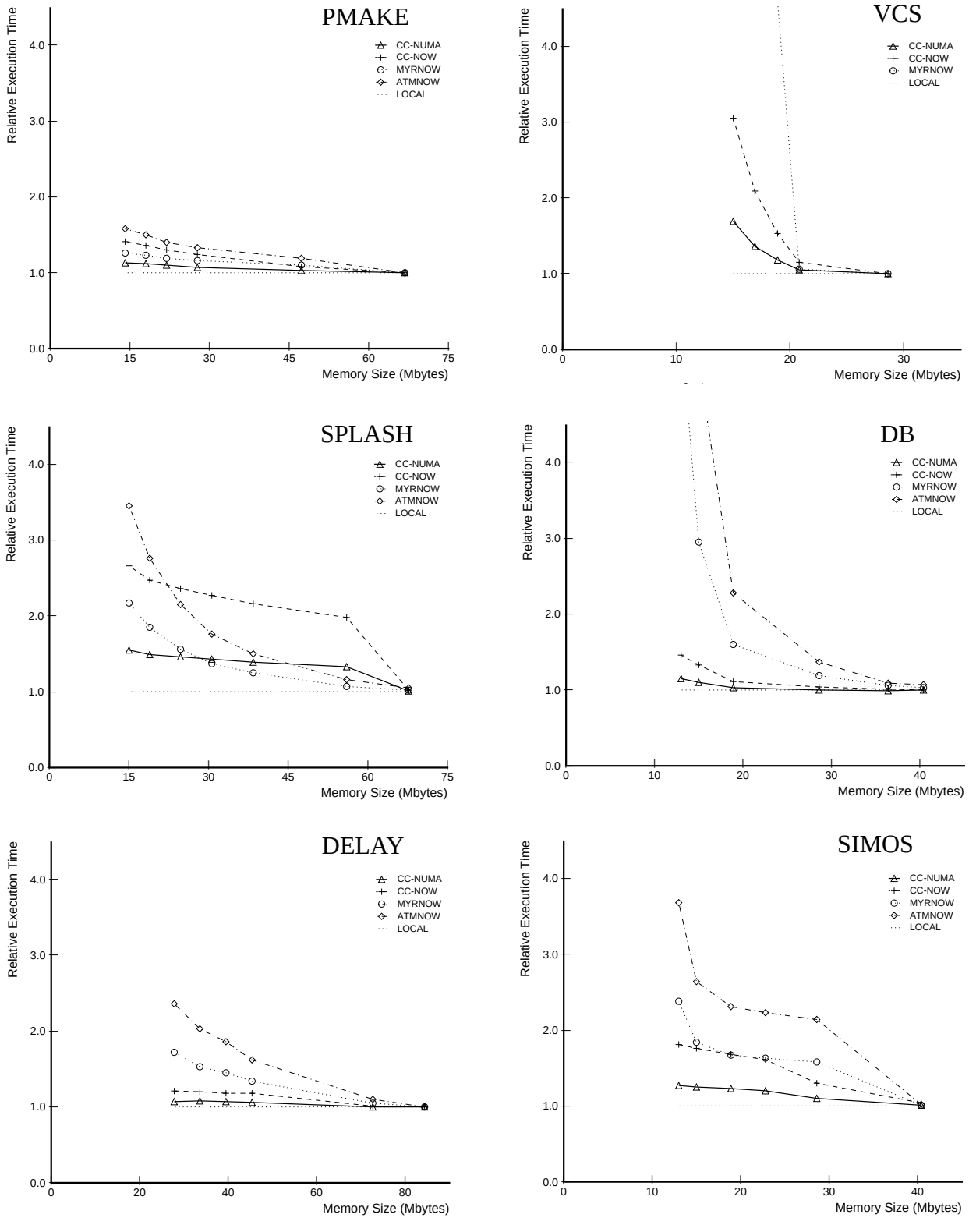

FIGURE 5. Relative execution time of the optimistic NOW models. Each graph shows execution time for different memory sizes, relative to the case with all memory local. There are four curves, the MYRNOW (MYRINET model), ATMNOW (ATM model), and the CC-NUMA, and CC-NOW models for comparison. We did not run the ATMNOW configuration for VCS.

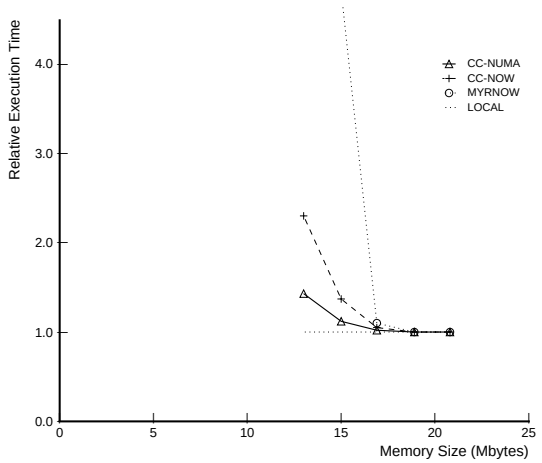

FIGURE 6. Performance of the MYRNOW model. We compare the performance of the MYRNOW model with that of the CC-NUMA models for a workload with only a single VCS application. It shows performance similar to the VCS workload

The comparison in performance between the MYRNOW and CC-NOW configurations is interesting. The cache-miss latency for accessing all the cache lines in a page from remote memory is about 100 microseconds for the CC-NOW case (3 microseconds per cache line and 32 cache lines in a page). This is the same as the page fault latency for the MYRNOW case, which in addition can also hide some of the latency by running another process.

For the DELAY and SIMOS workloads, the performance of the MYRNOW model is not better than that of CC-NOW. The smaller page-fault latency has helped, but unlike the previous two workloads, these are not multiprogrammed, so the page-fault latency cannot be hidden. However, their performance is still within the acceptable range because of the smaller page-fault latency.

For the VCS and DB workloads, even the MYRNOW architecture cannot deliver acceptable performance for smaller values of local memory. As explained in Section 4.1.1, these workloads have a sharp knee in the fault curve, and the high consequent fault rate and poor spatial locality cause the performance to degrade sharply and become unacceptable. The VCS workload is multiprogrammed with two jobs executing simultaneously. The results for a workload with only a single VCS job, shown in Figure 6, confirm that the performance is inherent to the memory access characteristics of the VCS application.

The ATMNOW model performs worse than the MYRNOW model for the workloads, though provides acceptable performance for some workloads, such as PMAKE and DELAY. The implication here is that the latency and network bandwidth of the MYRNOW will be needed to maintain good memory-sharing performance with a OS-based (NOW) model.

6.0 Other Performance Issues

Additional factors, current and future, can affect the performance of the memory-sharing models. Some of these are trends, such as advances in processor architectures, increases in processor speed, and increases in page size, while others are smarter software policies. We do a brief

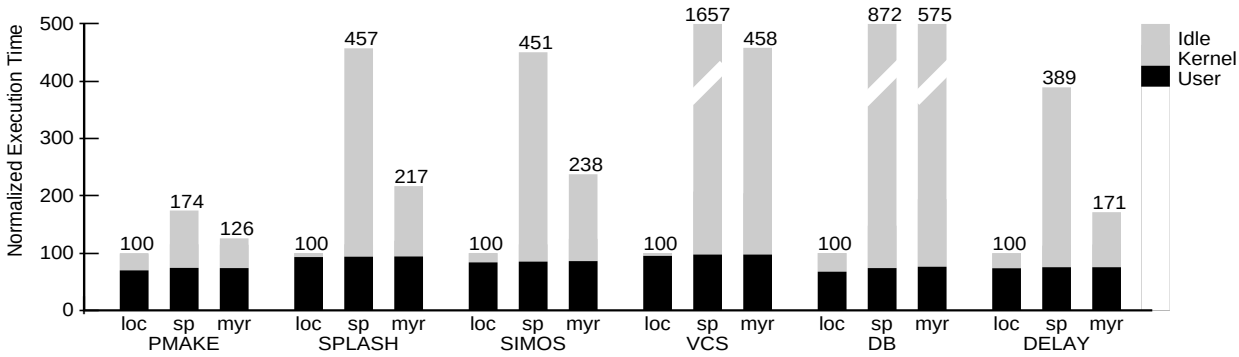

FIGURE 7. Breakdown of execution time for the SUBPAGE and MYRNOW models . Execution times are for the smallest memory size simulated, and are normalized to the all LOCAL case. The bars for the VCS and DB workloads overflow the figure.

analysis to see how some of these trends and policies might affect the performance results we have presented.

6.1 Processor Advances

We see two trends in processor architectures that are likely to affect the performance of memory-sharing models. The first is faster processors, and this is likely to benefit the software-based NOW model. The second is latency tolerating and hiding techniques, and these will benefit the hardware-based model.

Faster processors can help the performance of the software-based NOW model by reducing the time spent executing the server routines and the trap handlers on the client. By cranking down the page-fault latency, as in the MYRNOW model, we have already assumed significantly lower service times on the server, achieved through faster processors and other techniques. For the client-side trap handlers, the actual time spent in the kernel is only a small fraction of the total time for these models, most of the slowdown is due to idle time, as seen in Figure 7. For example, taking the case of the MYRNOW model for the worst performing workload, VCS, the kernel time is less than 25% of the total time. If the processor speed were to double (600 MHZ), the maximum reduction we would see in this case would be about 10%. This reduction would not bring the performance of the NOW model into the acceptable range for the VCS workload. The effect of increasing processor speed on the other workloads and the other NOW models would be even less than this, and so will not significantly change our results.

A number of features to hide or tolerate the latency of cache misses are being incorporated into processor architectures. These features, such as out-of-order execution, non-blocking caches, and prefetching, will improve the performance of the hardware-based model by reducing the stall time for remote access as seen by the workload. In fact, a multi-threaded processor attempts to hide cache-miss latency the same way that the OS-based model tries to hide page-fault latency, by executing from a different context [TEL95]. Increasing the size and associativity of the processor’s secondary cache will also help by reducing the number of cache misses seen.The OS-based model will not see any sharing benefit from these changes because every referenced page will have to be faulted into local memory.

6.2 Page Size

Another trend, increasing page size, primarily affects the performance of the OS-based model. The performance impact of larger page sizes would be a trade-off between the changes in the resulting fault rate (decided by the spatial locality of accesses to pages) and the bandwidth component of latency of a remote page fetch. The sub-page technique can be used to reduce the latter. A larger page-size would help workloads like PMAKE that have good spatial locality, but would hurt workloads like VCS and DB that exhibit more random accesses to pages.

6.3 OS-based policies

In our experiments we assumed a static first-touch placement of pages in the hardware-based model. This can result in a poor placement of pages, as we saw in the SPLASH workload, and cannot respond to temporal changes in the working set. The locality of data can be improved in the hardware model by dynamically migrating “hot” pages to local memory, thus improving performance [VDG+96]. While the OS-based model is forced to bring a referenced page to local memory, the OS in the hardware-based model can selectively migrate “hot” pages to local memory, thus avoiding the problem of thrashing that we see in some of the workloads. Therefore, a hardware-based model with an intelligent page migration policy, should always perform better than a pure OS-based model.

Performance of the NOW model could be improved by pre-fetching of pages, choosing intelligent page-replacement policies [PGG+95][FMP+95], and sub-page fetching [JFV+96]. The effect of these features are implementation and workload dependent.

7.0 Related Work

There has been little work comparing the performance of memory-sharing models for compute-server workloads. The only directly related work has been by Li & Petersen [LiP91]. Their work, while not done in the same context, addresses a similar question. They addressed the question of how to use additional slower memory in a computer system — either as slower extended memory directly accessible from the processor, or as a cache between the file system and main memory. They concluded that the former resulted in better performance. Our work differs on the following points. We frame the problem in the context of currently interesting systems, and the system used for evaluation has relevant modern characteristics, such as fast processors and large caches; their work was based on a SUN3/180. In addition to single application workloads, we consider multiprogrammed compute-server workloads as well. The latency of page faults can potentially be hidden in a multiprogrammed workload. Their runs, on an actual implementation, were restricted to a single set of latency parameters. Using SimOS, we are able to run the workloads for variations in each model, and explore the effect of changes in latency and interconnection bandwidth.

There has also been work comparing the performance of parallel applications on different implementations of software DSM systems on the NOW model, with that of the CC-NUMA model [JKW95][CDK+94][BZS93]. While the memory models being compared are the same, the nature of parallel applications and the issues considered are completely different from that of compute-server workloads of sequential applications.

8.0 Conclusions

Efficient sharing of memory resources in a cluster of workstations has the promise of greatly improving the performance and cost-effectiveness of the cluster when running large memory-intensive jobs. For memory-sharing to be effective, remote memory must be presented as an extension of local memory, both conceptually and through performance. Using SimOS we do a fair evaluation of the performance of different memory-sharing architectures using a variety of representative workloads.

With the increasing speed of commodity interconnection networks, we were interested in the possibility of achieving memory-sharing using a software-only solution. We analyzed the memory-sharing performance of such a solution, the OS-based NOW model that uses the mechanism developed for remote paging across a network to another workstation's memory. Based on our results for this model, if a significant part of a workload's data does not fit in local memory, the performance of workloads degrades to greater than two times slowdown. The latency of the remote page-fetch cannot be hidden by overlapping computation with communication for multiprogrammed workloads.

If in the future, the effective latency for a remote page fetch can be reduced from the current 0.5 - 1 millisecond to about 100 microseconds, the software-only model will be able to provide acceptable performance for many workloads. However, for the class of workloads that are characterized by mostly random accesses to memory, such as the commercial CAD simulator and the database, this form of memory-sharing is unsuitable and continues to show poor performance. Achieving this optimistic remote page-fault latency is likely to be difficult, requiring a memory-server response time of a few tens of microseconds, and a fast interconnect with latencies in the microseconds and bandwidth in the gigabits/sec range.

In contrast, the hardware-based model is able to consistently provide good stable memory-sharing performance. This is true for both environments of interest to us; the machine room type environment where the CC-NUMA model is comparable to the all-local-memory case, and the distributed user-workstation type environment where the CC-NOW model is within two times slowdown. Processor architecture trends and OS-based policies are likely to improve the performance of the hardware-based model. Our results lead to the conclusion that hardware support is required for efficient memory-sharing in workstation clusters when running memory-intensive jobs.

Bibliography

[ACD+91] A. Agarwal et al. The MIT Alewife Machine: A Large-Scale Distributed-Memory Multiprocessor. *MIT/LCS Memo TM-454, Massachusetts Institute of Technology,* 1991.

[ACP+94] T. Anderson, D. Culler, D. Patterson. A Case for NOW (Networks of Workstations). Presented at *Principles of Distributed Computing,* August 1994.

[ADN+95] T. Anderson et al. Serverless Network File Systems. In *Proceedings of the Fifteenth ACM Symposium on Operating Systems Principles,* pages 109-126, December 1995.

[Arp+95] R.H. Arpaci et al. The interaction of parallel and sequential workloads on a network of workstations. In *Proceedings of 1995 ACM SIGMETRICS Joint International*

Conference on Measurement and Modeling of Computer Systems, pages 267-278, May 1995.

[Bod+95] N. J. Boden et al. MYRINET: A GIGABIT PER SECOND LOCAL AREA NETWORK. *IEEE-Micro,Vol.15, No.1,* pages 29-36, February 1995.

[BZS93] B. Bershad, M. Zekauskas, W. Sawdon. The Midway distributed shared memory system. *COMPCON Digest of Papers*, pages 528-537, Feb. 1993.

[CDK+94] A. Cox et al. Software versus hardware shared-memory implementation: a case study. In *Proceedings of the 21st Annual International Symposium on Computer Architecture,* pages 106-17, April 1994.

[DWA+94] M. Dahlin, R. Wang, T. Anderson, D. Patterson. Cooperative Caching: Using Remote Client Memory to Improve File System Performance. In *Proceedings of the First Symposium on Operating Systems Design and Implementation*, pages 184-190, March 1989.

[FMP+95] M. Feeley, W. Morgan, F. Pighin, A. Karlin, H. Levy, C. Thekkath. Implementing Global Memory Management in a Workstation Cluster. In *Proceedings of the Fifteenth ACM Symposium on Operating Systems Principles*, pages 201-212, December 1995.

[JFV+96] H. Jamrozik et al. Reducing Network Latency Using Subpages in a Global Memory Environment. In *Proceedings, Architectural Support for Programming Languages and Operating Systems*, pages 258-267, October 1996.

[JKW95] K. Johnson, F. Kaashoek, D. Wallach. CRL: High-Performance All-Software Distributed Shared Memory. In *Proceedings of the Fifteenth ACM Symposium on Operating Systems Principles*, pages 213-228, December 1995.

[Kus+94] J. Kuskin, et al. The Stanford FLASH Multiprocessor. In *Proceedings of the 21st International Symposium on Computer Architecture*, pages 302-313, April 1994.

[LLG+92] D. Lenoski et al.The Stanford DASH Multiprocessor. *IEEE Computer 25(3),* pages 63-79, March 1992.

[LiP91] K. Li and K. Petersen. Evaluation of Memory System Extensions. In *Proceedings of the 18th International Symposium on Computer Architecture,* pages 84-93, May1991.

[NAB+94] A. Nowatzyk, G. Aybay, M. Browne, E. Kelly, D. Lee, and M. Park. “The S3.mp Scalable Shared Memory Multiprocessor.” In *Proceedings of 27th Hawaii International Conference on Systems Sciences*, pages. 144-153, January 1994

[PGG+95] R. H. Patterson et al. Informed Prefetching and Caching. In *Proceedings of the Fifteenth ACM Symposium on Operating Systems Principles*, pages 79-95, December 1995.

[RHW+95] M. Rosenblum, S. Herrod, E. Witchel, and A. Gupta. Complete Computer Simulation: the SimOS approach. In *IEEE Parallel and Distributed Technology,* Fall 1995.

[RLW94] S. Reinhardt, J. Larus, D. Wood. Typhoon and Tempest: User-Level Shared Memory. *ACM/IEEE International Symposium on Computer Architecture (ISCA),* pages 392-403, June 1995.

[TEL95] D Tullsen, S Eggers, H Levy. Simultaneous multithreading: Maximizing on-chip parallelism. *In Proceeding International Symposium on Computer Architecture (ISCA),* April 1994).

[VDG+] B Verghese, S Devine, A Gupta, and M Rosenblum. Operating System Support for Improving Data Locality on CC-NUMA Compute Servers. In *Proceedings, Architectural Support for Programming Languages and Operating Systems*, pages 279-289, October 1996.