Using Remote Memory to avoid Disk Thrashing: A Simulation Study

Evangelos P. Markatos[*]
Computer Architecture and VLSI Systems Group
Institute of Computer Science (ICS)
Foundation for Research & Technology – Hellas (FORTH)
Vassilika Vouton, P.O. Box 1385, GR 711 10 Heraklion, Crete, Greece

In Proceedings of the MASCOTS'96, San Jose, CA, USA, Feb. 1996

Abstract

The increasing use of high-bandwidth and low-latency networks make possible the use of remote (network) memory as an alternative to disk means of storing an application's data, because remote-to-local memory transfers over a modern interconnection network are faster than traditional disk-to-memory transfers. In this paper we explore the possibility of using the remote memory as (i) a (faster-than-disk) backing store, (ii) an extension of main memory accessed using single (remote) memory references, and (iii) as a combination of both. We use execution driven simulation to investigate the performance impact the use of remote memory has on several real programs. We conclude that even for today's low throughput networks, using remote memory as a place for storing (some) of an application's data may result in significant performance improvements, which will continue to get higher, as the disparity between disk transfer rates and network transfer rates continues to increase.

1 Introduction

Recent applications like multimedia, windowing systems, scientific computations, engineering simulations, etc. running on workstation clusters (or network of PCs) require an everincreasing amount of memory. Although the *total* amount of main memory in a workstation cluster is usually sufficient for these applications, the amount of physical memory provided by each *individual* workstation is usually insufficient to cover the needs of current and near future applications. To make matters worse, the use of *multiprogramming* and *time-sharing* reduces the amount of *physical* main memory available to each application even more. To keep the workstation costs low, most workstations usually have a powerful CPU, but they do not have the amount of main memory that some applications need. Thereby, applications run efficiently as long as their working set fits in the main memory of the workstation. As soon as the working set exceeds the main memory size, the performance of the application suffers severely.

We believe that applications who need more main memory than a single workstation can provide, should make use of the remote main memory[1] available in a workstation cluster, provided that the appropriate system software support exists. In this paper we explore the trends and circumstances under which the use of remote memory for storing an applications data is beneficial from a performance point of view. Recently, there have been two recent architecture developments that make the use of remote memories attractive:

- Memory-to-memory transfer rates between workstations have increased sharply in the last few years, due to corresponding increases in local area network throughput
- Modern architectures provide low-latency remote-memory accesses, in order to provide an efficient mechanism for fast message-passing and shared-memory systems [4, 14]

In this paper we show that it is performance-wise to use remote memory to efficiently store and retrieve an application's data in workstation clusters. In section 3 we describe several methods with which applications can make use of the available remote memory: (i) as a fast RAM disk, (ii) as a (slow) main memory accessed via regular read and write operations, and (iii) as a combination of both. In order to evaluate the various architecture alternatives and the various paging methods build on top of them, we use execution-driven simulation, and present our performance results. Section 4 places our work in the proper context by comparing it with related work. Finally, section 5 discusses extensions of our approach, and presents our major conclusions.

2 Remote Main Memory

The architecture we assume is a distributed system composed of workstations and an interconnection network.

[*]E.P. Markatos is also with the University of Crete. He can be reached at markatos@csi.forth.gr

[1]In a network of workstations each workstation has a processor and its local memory. The main memory of all other workstation is referred to as the *remote* memory (also called network memory [2]). The same definition can be extended to a network of PCs and to a parallel computer.

Each workstation consists of a processor and its local memory. The memory of all the other workstations in the system comprise the remote memory. In the simplest version of this distributed system (the DISK), the remote memory is not used. The only paging device is the disk. When a page needs to be evicted from local memory, it is sent to disk; when a page needs to be fetched to local memory, it is fetched from disk. This is exactly what most paging systems do today.

To facilitate fast page transfers, remote memory could be used as a paging device (RAM_DISK). When a page must be paged-out to make room for newly referenced data, the pager sends the page to remote memory. When the page is later needed, it is fetched from remote memory. RAM_DISK allows only whole page transfers at a time. Both RAM_DISK and DISK experience the same number of page faults, but RAM_DISK does not suffer from seek and rotational latencies as the disk does.

In our next version of the architecture, the REMOTE_MAIN, remote memory is used as main memory accessed using single read and write operations, while the disk is used as the backing store. The pager works as follows: When a page is referenced, it is brought into local main memory if there is an available frame. Otherwise, it is sent to remote main memory. If neither part of main memory has an available frame, a page residing in local memory is evicted to disk to make room for the new page. Once a page has been mapped into an (either local or remote) frame, it is never moved (unless both local and remote memories fill up). Both local and remote memory are accessed via regular load and store operations.

In the last version of the architecture, the COUNTERS, we assume the existence of a reference counter per remote page. When a processor accesses a remote page, the counter associated with the page is decremented. When the counter reaches zero, an interrupt is sent to the operating system, which replicates the page locally, sending a local page to remote memory to make available space. This architecture is a combination of RAM_DISK and REMOTE_MAIN, and we believe that combines the advantages of both. When a page is frequently accessed, COUNTERS migrate the page to local memory, much like RAM_DISK, thus providing fast access to the page. When a page in infrequently accessed, COUNTERS provide remote access to the page, much like REMOTE_MAIN, thus avoiding unnecessary page transfers.

3 Experiments

In this section we evaluate the performance of various remote memory configurations, and of various paging policies. We use ATOM [16], an object code rewriting tool that gathers traces of applications, and simulates a memory system on-the-fly. ATOM takes as input an executable and instruments it with calls to simulation routines. The application is actually executed, while at the same time, our simulator runs to simulate the paging architecture we want. The parameters of the architecture simulated can be found in table 1.

parameter	cost (in cycles)
local memory access	1 cycle
remote memory access	100 cycles
network bandwidth	25 Mbytes / sec
page size	8 Kbytes
local memory size	8 Mbytes
remote memory size	80 Mbytes
operating systems overhead	500 cycles
disk transfer rate	5 Mbytes/sec
disk seek latency	2.5 ms

Table 1: Architecture Parameters.

3.1 Simulation Environment

3.1.1 The Applications

We have chosen the applications from several domains (numerical analysis, matrix operations, data management, and circuit simulation) so that they represent a wide variety of data access patterns. We use the following applications:

- MVEC: a matrix-vector multiplication; matrix size 1500×1500 elements.
- GAUSS: A gauss elimination on a 1500×1500 matrix.
- SORT: A sorting application of 1400 records, of 8Kbytes each, using the UNIX provided qsort library routine.
- TRANS: Transposing a 1500×1500 matrix. This application transposes an input matrix in place. It is an example of application with almost no locality at all. Both rows and columns of the matrix are accessed at the same time, putting a tremendous stress into the paging system. We expect the traditional paging system to thrash, while the use or remote memory in storing and retrieving data will significantly improve the performance of the system.
- VERILOG: Verilog is a hardware simulation language. In this example, we simulate the Telegraphos switch which has been designed by members of our Lab [11].

The working set and trace length of each application are described in the following table:

Application	Number of references in millions	Working set in Mbytes
TRANS	22.6	24
MVEC	25	23
GAUSS	82.9	22
SORT	142.4	14
VERILOG	100	26.7

In all our simulation studies the paging policy is LRU. We have also experimented with a random page replacement policy, and have seen similar results.

3.2 Results

3.2.1 Remote Memory vs. Disk

The first question we set out to answer is the usefulness of remote memory for storing an application's data. We executed the applications on a DEC Alpha workstation and simulated the paging architectures as described. For each

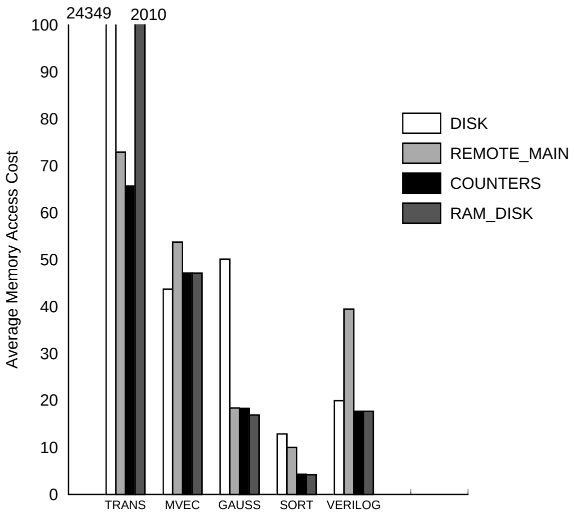

Figure 1: **Paging policies**. *Performance of our application suite under various paging configurations. In almost all cases* `COUNTERS` *has the best performance, followed by* `RAM_DISK`.

application/architecture combination we measured the average memory access cost of the application running on the architecture, and plotted the results in figure 1. We see that in three out of five applications (`TRANS`, `GAUSS` and `SORT`) `DISK` performs worse than the other architectures, and sometimes, substantially so. In `TRANS` for example, where the is no locality at all, the `DISK` policy performs $24349/65.65 = 371$ times worse than the `COUNTERS` policy. The inferiority of the `DISK` policy over `COUNTERS` is obvious (but not as pronounced as in `TRANS`) in `GAUSS` and `SORT` applications as well. The only exception is the `MVEC` application in which `DISK` is better than counters by about 10%. The reason is that `MVEC` reads (most of) its input only once and never uses it again; thus, any sophisticated paging policies do not improve performance [2].

Among the policies that make use of remote memory, `COUNTERS` seems to be the best (or close to the best) in all cases. `RAM_DISK` is better than `REMOTE_MAIN` with the exception of the `TRANS` application, where the system serviced an extraordinary amount of page faults. `REMOTE_MAIN` is almost always the worst policy, because the remote memory access cost is large (100 to 1) over the local memory access cost. Li and Petersen [12] have done experiments on a system where the remote memory access cost was only twice the local memory access cost, and in their system, `REMOTE_MAIN` was much more attractive than `RAM_DISK`. Unfortunately, as architecture trends

[2]The reader may notice that all policies have a somewhat high average memory access cost. This is because we do not assume multiprogramming; thus when an application page faults, the whole system waits for the fault to be serviced and the page to arrive (possibly from the disk). It seems that multiprogramming could hide some of the disk latency, but in our set of experiments, multiprogramming would reduce the amount of *physical* main memory available to each application, thus leading to more page faults, and to even worse performance.

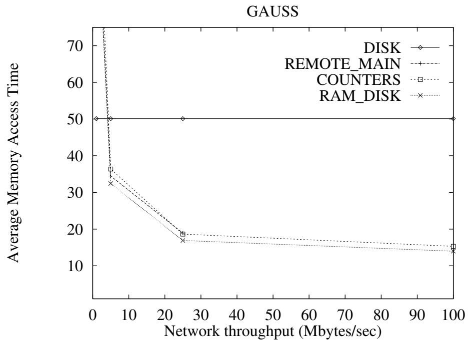

Figure 2: **Performance of paging policies as a function of Network throughput:** `GAUSS` **application.** *The performance of* `DISK` *is insensitive to network throughput. All other policies improve with network throughput. It is interesting to note that even when network throughput is as low as the disk transfer rate (5 Mbytes/sec),* `DISK` *is still inferior to remote memory paging policies.*

suggest, the cost of remote memory access will continue to increase compared to local memory access cost, [9], and thus we expect the usefulness of `REMOTE_MAIN` to decrease. However, we should note that the ability to make single remote memory accesses is valuable when used prudently, as in the `COUNTERS` policy.

3.2.2 The Influence of Network Throughput

Storing an application's data in remote memory hasn't been popular so far, partly because local area networks did not provide high throughput, making memory-to-memory transfer over a LAN comparable to a disk-to-memory transfer over the local disk. To evaluate the effect the network throughput has on remote paging policies, we simulated the execution of our applications and varied the interconnection network throughput from 1Mbyte/sec to 100 Mbytes/sec. The results for `GAUSS` are shown in figure 2 [3]. When the network throughput is low (1 Mbyte/sec), the `DISK` outperforms all remote memory paging policies because the disk-to-memory transfer is 5 Mbytes/sec, thus disk-to-memory transfers are (almost) 5 time faster than memory-to-memory transfers over the network. However, when the network throughput becomes equal to the disk-to-memory throughput (5 Mbytes/sec), both `COUNTERS` and `RAM_DISK` outperform the simple `DISK` policy; even when the network throughput is the same as the disk throughput it is more efficient to use the remote memory instead of the disk, as backing store! The reason is simple: remote memory does not suffer from seek and rotational delays as the disk does. When network throughput is as high as 100 Mbytes/sec, the DISK is clearly inferior to all remote paging policies. Even though when the network throughput is moderately high (close to 25 Mbytes/sec), the performance of remote memory paging policies is clearly higher than that of the `DISK`.

[3]Due to space constraints we do not show the results for the other applications which are similar.

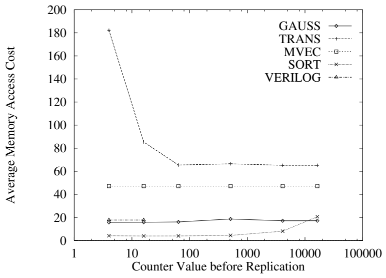

Figure 3: **Tuning the** COUNTERS **policy.** *When the initial value of the counter is between 64 and 512, all applications perform close to their best.*

3.2.3 Tuning the COUNTERS policy

The COUNTERS policy does not bring a page to local memory on its first access, like the RAM_DISK and the DISK policy. Instead, it maps the page remotely, and allows the processor to make remote memory accesses to it. Only after the processor has make a sufficient number of remote accesses, then the page is migrated locally. This number is set by the operating system on a per page basis. If the number is very low, COUNTERS behaves much like RAM_DISK, bringing pages locally (almost) on the first access. When the number is very high, COUNTERS behaves much like REMOTE_MAIN, (almost) never replicating pages. When the performance of REMOTE_MAIN is sufficiently different from RAM_DISK, then adjusting the initial value of the counter, makes COUNTERS behave like the best policy. In this set of experiments we investigate the influence of the initial value of the counter on the performance of the COUNTERS policy. To understand the influence of the initial value of the counter has on the COUNTERS policy, we simulated the execution of our applications varying the initial value of the counter for each page and plotted the result in figure 3. We see that the performance of TRANS application improves with the counter value. The reason is that TRANS has almost no locality at all. That is, the processor will make only a few accesses to each page, before paging it out, and thus, it is not worth it to bring the page locally. Thus, the higher the counter value, the better the performance of the application is. Exactly the opposite holds for SORT, whose performance get worse with higher counter values. The reason is that the sorting application makes several accesses to each page before the page is paged out, thus, it is worthwhile to bring pages in the local memory as soon as possible. For MVEC and GAUSS the initial value of the counter makes little difference in performance because both REMOTE_MAIN and RAM_DISK perform close to each other.

By observing closely figure 3 we note that for all applications when the counter value is between 64 and 512, the performance of the COUNTERS policy is (almost) the best. Thus, we see that the COUNTERS policy does not require difficult tuning. An initial value between 64 and 512 for each counter is a reasonable choice.

3.2.4 Simulating Remote Memory Accesses in Software

Although providing single remote memory accesses is helpful to avoid thrashing (like the TRANS application in

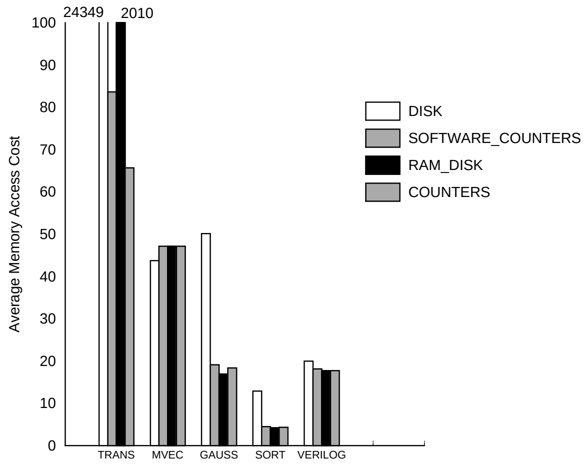

Figure 4: **Paging policies.** *The effectiveness of simulating counters in software (*SOFTWARE_COUNTERS*) is evaluated. In most cases,* SOFTWARE_COUNTERS *perform very well, always close to hardware* COUNTERS*.*

figure 1), few architectures provide hardware implemented remote memory accesses. Several systems though, implement them completely, or mostly, in software at reasonable performance cost. To investigate the usefulness of software implemented remote memory accesses we simulated the performance of our applications running under the COUNTERS policy using software implemented counters (called SOFTWARE_COUNTERS). [4] The performance of the SOFTWARE_COUNTERS compared to the known DISK, COUNTERS, and RAM_DISK policies can be found in figure 4. We see that the performance of SOFTWARE_COUNTERS is only slightly worse than the performance of hardware provided COUNTERS. Moreover, SOFTWARE_COUNTERS perform significantly better than RAM_DISK for applications with no locality (e.g. TRANS), while for the other applications that have good locality, SOFTWARE_COUNTERS performs comparable to RAM_DISK, and COUNTERS.

4 Related Work

This paper evaluates the performance benefits of using remote memory as a place to store an application's data. Its main contributions are (i) the proposal of a new policy (COUNTERS) that uses remote memory both as main memory and as backing store, (ii) the evaluation of memory management policies for *real* applications, and (iii) the systematic exploration of various parameters that influence the use of remote memory for storing an application's data.

Li and Petersen [12] have implemented a related system where they add main memory on the I/O bus (VME bus) of a computer system. This memory can be used both as backing store, but also as (slow) main memory accessed via regular load and store operations.

Several research groups study the issues in using remote memory in a workstation cluster to improve paging perfor-

[4] In SOFTWARE_COUNTERS the remote memory access cost is 1000 cycles.

mance [2, 3, 5, 8, 10, 13, 15] and file system performance [1, 6, 7]. Our work differs from previous approaches in two aspects: (i) we provide extensive performance results based on *execution-driven simulation* of real applications, while previous approaches have provided very limited performance results (at least with respect to paging), and (ii) we explore the use of *single* remote memory references as a mechanism to access infrequently used pages by proposing a new policy called `COUNTERS`. This policy clearly outperforms all previously proposed policies.

5 Conclusions

In this paper we present a memory shortage problem faced by several applications that run on a tightly-coupled distributed system, or multiprocessor system, and explain how to use remote memories to store an applications' data. We use execution-driven simulation of real applications to evaluate the usefulness remote memory. We have shown that in most cases, using remote memory for storing an application's data is faster than the disk, sometimes substantially so, especially when the disk suffers from thrashing.

Based on our experiments we conclude that:

- Using remote memory instead of the disk to store an application's data that do not fit in local main memory has shown to improve performance over disk for several applications and several ranges of parameters.
- Even when the network throughput is as low as the disk transfer rate, using remote memory to store an application's data may be better than using the disk, because remote memory does not suffer from seek and rotational delay as the disks do.
- The `COUNTERS` policy that uses remote memory *both* as (slow) main memory *and* as a (fast) backing store at the same time has shown a stable performance; always best, or close to the best because it combines quick page migration for frequently used pages, and avoids unnecessary replications for infrequently used pages.

Based on our conclusions we believe that storing an application's data on remote memory (instead of the local disk), in several cases results in performance improvements, which will increase in the near future, if current architecture trends continue to hold.

Acknowledgments

This work was developed in the ESPRIT/HPCN project "SHIPS", and will form a test application for the project "ARCHES", funded by the European Union. We deeply appreciate this financial support, without which this work would have not existed.

We would like to thank Catherine Chronaki for useful comments in earlier drafts of this paper.

References

[1] T. E. Anderson, M. D. Dahlin, J. M. Neefe, D. A. Patterson, D. S. Roselli, and R. Y. Wang. Serverless Network File Systems. In *Proc. 15-th Symposium on Operating Systems Principles*, December 1995.

[2] Thomas E. Anderson, David E. Culler, and David A. Patterson. A Case for NOW (Networks of Workstations). *IEEE Micro*, 15(1):54–64, February 1995.

[3] G. Bernard and S. Hamma. Remote Memory Paging in Networks of Workstations. In *Proceedings of the SUUG International Conference on Open Systems: Solutions for Open Word*, April 1994.

[4] M. Blumrich, K. Li, R. Alpert, C. Dubnicki, E. Felten, and J. Sandberg. Virtual Memory Mapped Network Interface for the SHRIMP Multicomputer. In *Proceedings of the Twenty-First Int. Symposium on Computer Architecture*, pages 142–153, Chicago, IL, April 1994.

[5] D. Comer and J. Griffoen. A new design for Distributed Systems: the Remote Memory Model. In *Proceedings of the USENIX Summer Conference*, pages 127–135, 1990.

[6] T. Cortes, S. Girona, and J. Labarta. PACA: A Distributed File System Cache for Parallel MAchines. Performance under Unix-like workload. Technical Report UPC-DAC-1995-20, Departament d'Arquitectura de computadors, Universitat Politecnica de Catalunya (UPC), June 15 1995.

[7] M. J. Feeley, W. E. Morgan, F. H. Pighin, A. R. Karlin, H. M. Levy, and C. A. Thekkath. Implementing Global Memory Management in a Workstation Cluster. In *Proc. 15-th Symposium on Operating Systems Principles*, December 1995.

[8] E. W. Felten and J. Zahorjan. Issues in the Implementation of a Remote Memory Paging System. Technical Report 91-03-09, University of Washington, November 1991.

[9] J. L. Hennessy and D. A. Patterson. *Computer Architecture: A Quantitative Approach*. Morgan Kaufmann Publishers, Inc., 1990.

[10] L. Iftode, K. Li, and K. Petersen. Memory Servers for Multicomputers. In *Proceedings of COMPCON 93*, 1993.

[11] M. Katevenis, P. Vatsolaki, and A. Efthymiou. Pipelined Memory Shared Buffer for VLSI Switches. In *Proceedings of the ACM SIGCOMM '95 Conference*, pages 39–48, August 1995.

[12] K. Li and K. Petersen. Evaluation of Memory System Extensions. In *Proc. 18-th International Symposium on Comp. Arch.*, pages 84–93, 1991.

[13] E.P. Markatos and G. Dramitinos. Implementation of a Reliable Remote Memory Pager. In *Proceedings of the Usenix Technical Conference*, January 1996.

[14] E.P. Markatos and M. Katevenis. Telegraphos: High-Performance Networking for Parallel Processing on Workstation Clusters. In *Proceedings of the Second International Symposium on High-Performance Computer Architecture*, February 1996.

[15] B.N. Schilit and D. Duchamp. Adaptive Remote Paging for Mobile Computers. Technical Report CUCS-004-91, University of Columbia, 1991.

[16] Amitabh Srivastava and Alan Eustace. ATOM: A System for Building Customized Program Analysis Tools. In *Proceedings of the SIGPLAN '94 Conference on Programming Language Design and Implementation*, Orlando, FL, June 1994.